Multi-Key Homomorphic Secret Sharing

From Theory To Practice

Multi-Key Homomorphic Secret Sharing

Geoffroy Couteau, **Lali Devadas**, Aditya Hegde, Abhishek Jain, Sacha Servan-Schreiber

Roadmap

1. Summary of our contributions
    a. Motivating example: two-party succinct secure computation
    b. Define multi-key homomorphic secret sharing (MKHSS)
    c. Application: non-interactive conditional key exchange
2. Background on HSS from DCR
3. Constructing MKHSS from the DCR assumption

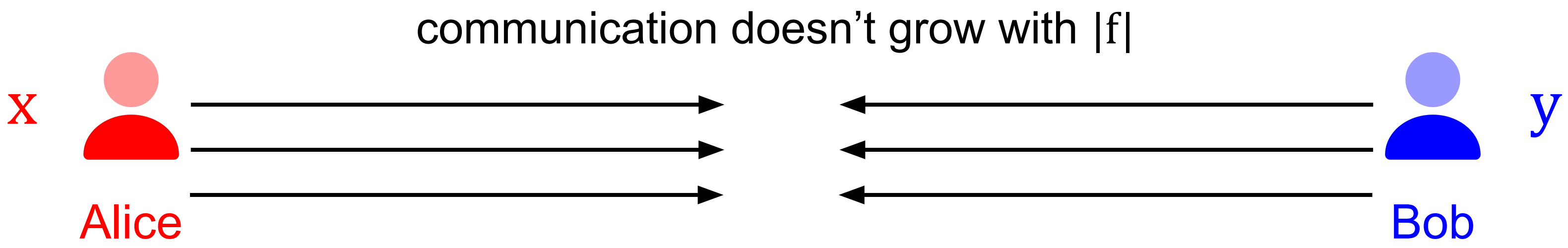
next: Kevin will talk about optimized implementations of MKHSS/key exchange

Roadmap

1. Summary of our contributions
   a. Motivating example: two-party succinct secure computation
   b. Define multi-key homomorphic secret sharing (MKHSS)
   c. Application: non-interactive conditional key exchange
2. Background on HSS from DCR
3. Constructing MKHSS from the DCR assumption

next: Kevin will talk about optimized implementations of MKHSS/key exchange

Motivation: Two-Party Succinct Secure Computation



$f(x,y)$

$f(x,y)$

learns nothing else about y

learns nothing else about x

prior to this work:

- **MKFHE**: 2 rounds from lattices
- **HSS**: 3 rounds from groups for NC1 circuits f

this work [CDHJS'25]:

- **MKHSS**: 2 rounds from groups for NC1 circuits f

Multi-Key HSS [CDHJS'25]

CRS

x Alice

Bob y

semantic security: hides x

$pe_A^{(x)} \leftarrow \mathrm{Encode}(x)$

$$pe_A^{(x)} \longrightarrow \quad \longleftarrow pe_B^{(y)}$$

$pe_B^{(y)} \leftarrow \mathrm{Encode}(y)$

$k_A \leftarrow \mathrm{Eval}_A(f, pe_B^{(y)})$

$$k_A \longrightarrow \quad \longleftarrow k_B$$

$k_B \leftarrow \mathrm{Eval}_B(f, pe_A^{(x)})$

$k_A - k_B = f(x,y)$

correctness

$k_A - k_B = f(x,y)$

Our results [CDHJS'25]

we construct multi-key HSS for NC1 circuits from any of the following:

- Decisional Diffie-Hellman (DDH)
- DDH-like assumptions over class groups
- Decisional Composite Residuosity (DCR)

this talk

this the first two-round succinct secure computation protocol from group-based assumptions for NC1 circuits.

Applications [CDHJS'25]

MKHSS achieves our goal of two-round succinct secure computation.

Q: after exchanging simultaneous messages, Alice and Bob have subtractive shares of the output – are there applications where this is sufficient?

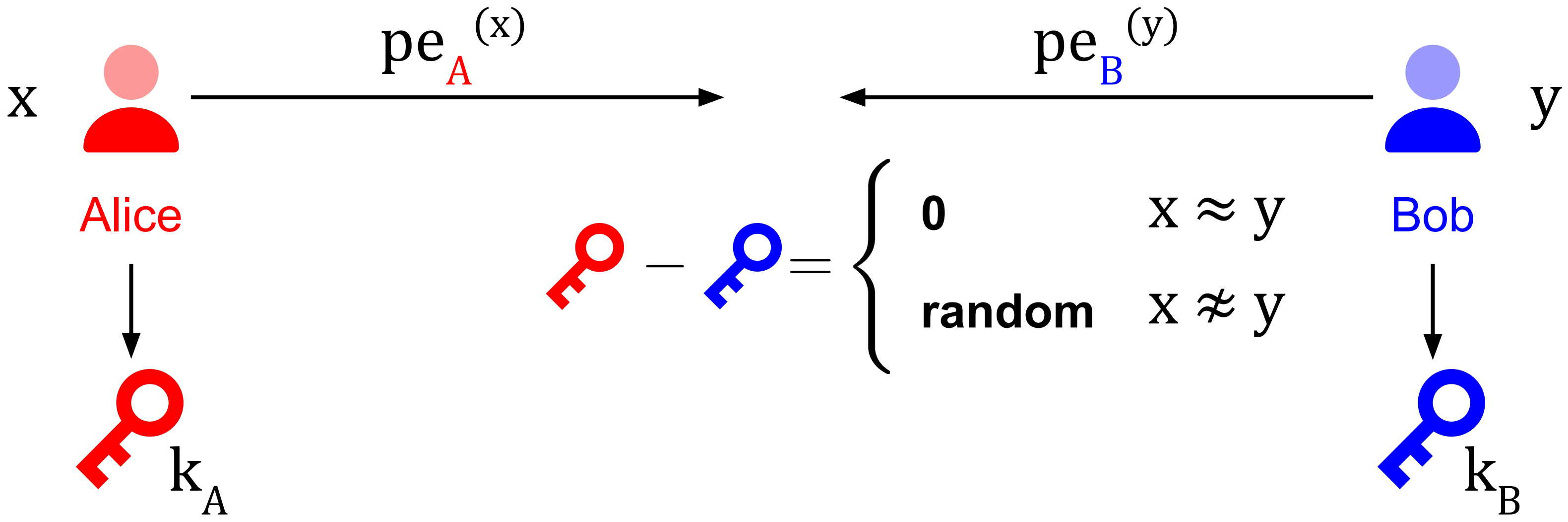
A: yes!

subtractive structure of shares also gives interesting *non-interactive* applications

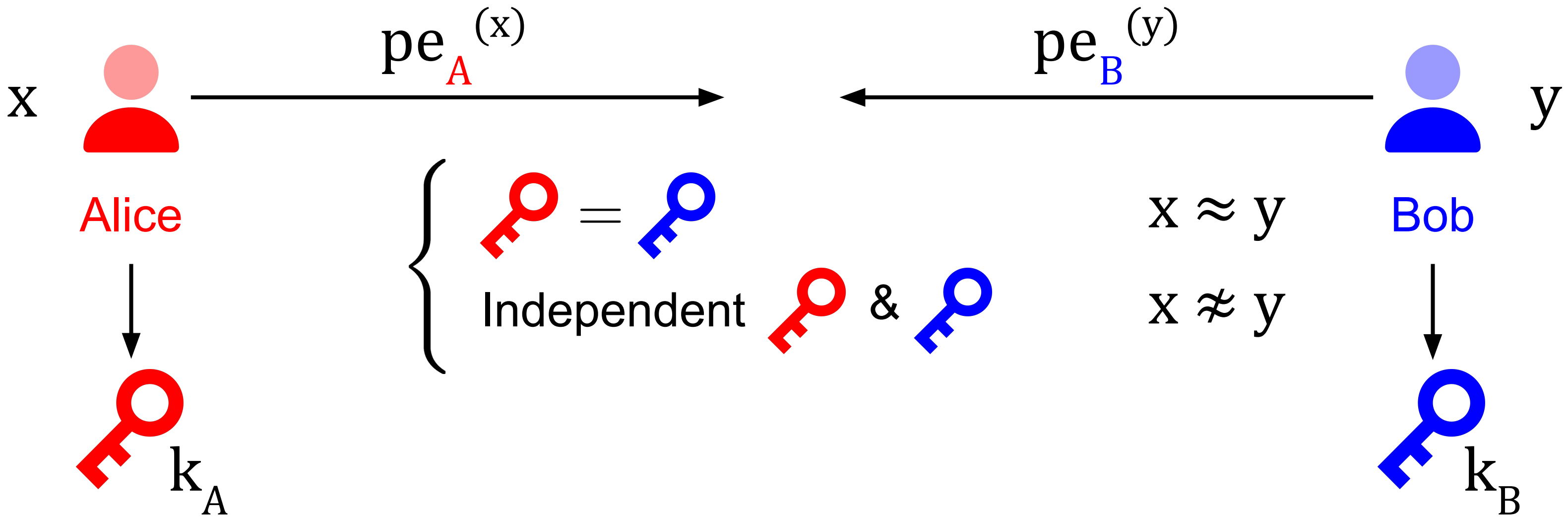
- non-interactive conditional key exchange
- public-key pseudorandom correlation functions
- silent preprocessing secure computation

this talk

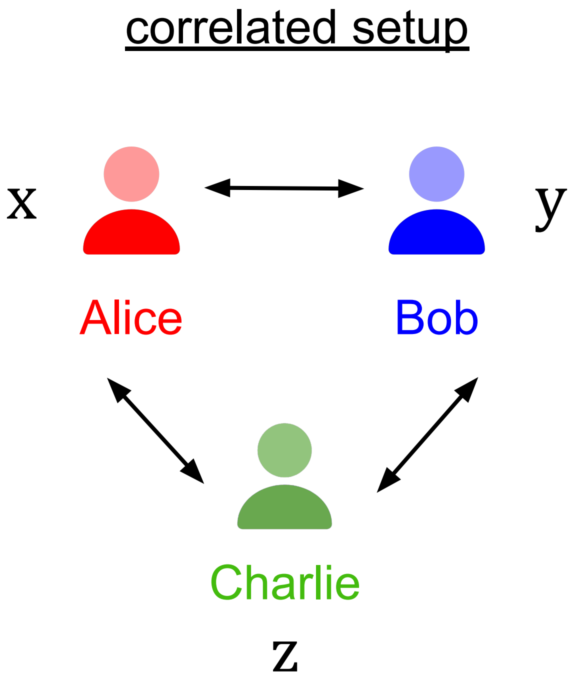
Application: Non-interactive Conditional Key Exchange

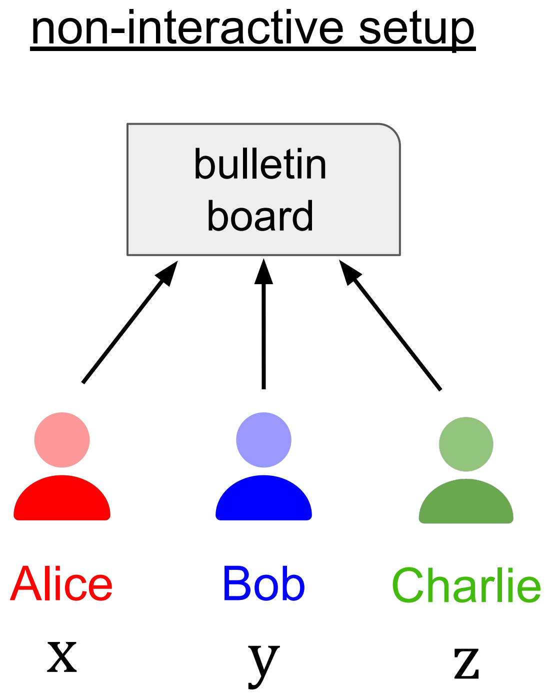


Application: Non-interactive Conditional Key Exchange



Added Benefit: Reusability





Roadmap

1. Summary of our contributions
   a. Motivating example: two-party succinct secure computation
   b. Define multi-key homomorphic secret sharing (MKHSS)
   c. Application: non-interactive conditional key exchange
2. Background on HSS from DCR
3. Constructing MKHSS from the DCR assumption

next: Kevin will talk about optimized implementations of MKHSS/key exchange

HSS template [Boyle-Gilboa-Ishai'16]

we make this setup non-interactive

0. assume that Alice and Bob have pk and shares of sk
1. they exchange encodings of their inputs x and y under pk
2. they perform local computations to obtain shares of output $f(x,y)$

next we will see how *input encodings* and *local computations* work for HSS from DCR [OSY'21] (with some helpful modifications)

Input encodings: Paillier-ElGamal encryptions

N = product of two safe primes
g = generator of the $2N^{th}$ residue subgroup of $Z^*_{N^2}$

$$\text{Input\_Encode}(pk, x) = (\, \text{Enc}_{pk}(x \cdot sk),\ \text{Enc}_{pk}(x)\,)$$

for HSS from DCR, these are Paillier-ElGamal encryptions [BCP'03]

$$sk \leftarrow\$ [N] \qquad pk = g^{-sk} \bmod N^2 \qquad \text{Enc}_{pk}(x) = (g^r \bmod N^2, pk^r (1+N)^x \bmod N^2)$$

we will use a "flipped encryption" for the other component

$$\text{Enc}_{pk}(x \cdot sk) = (g^r (1+N)^x \bmod N^2, pk^r \bmod N^2)$$

(this helps us later because it can be computed without knowing sk)

Input encodings: Paillier-ElGamal encryptions

another helpful note for later:

$$pk = g^{-sk} \qquad\qquad \mathrm{Enc}_{pk}(x) = (g^r, pk^r\,(1+N)^x)$$

what happens if we do $(pk^r\,(1+N)^x)^{sk'}$ for some sk'?

we end up with a ciphertext of $x \cdot sk'$ with respect to public key $pk^{sk'}$:

$$(g^r, (pk^{sk'})^r\,((1+N)^x)^{sk'}) = (g^r, (g^{sk \cdot sk'})^r\,(1+N)^{x \cdot sk'})$$

also a ciphertext which decrypts to $x \cdot sk'$ using secret key $sk \cdot sk'$.

morally multiplying message and secret key by same value sk'

Local computations: RMS multiplication

our HSS supports evaluating RMS multiplication programs:

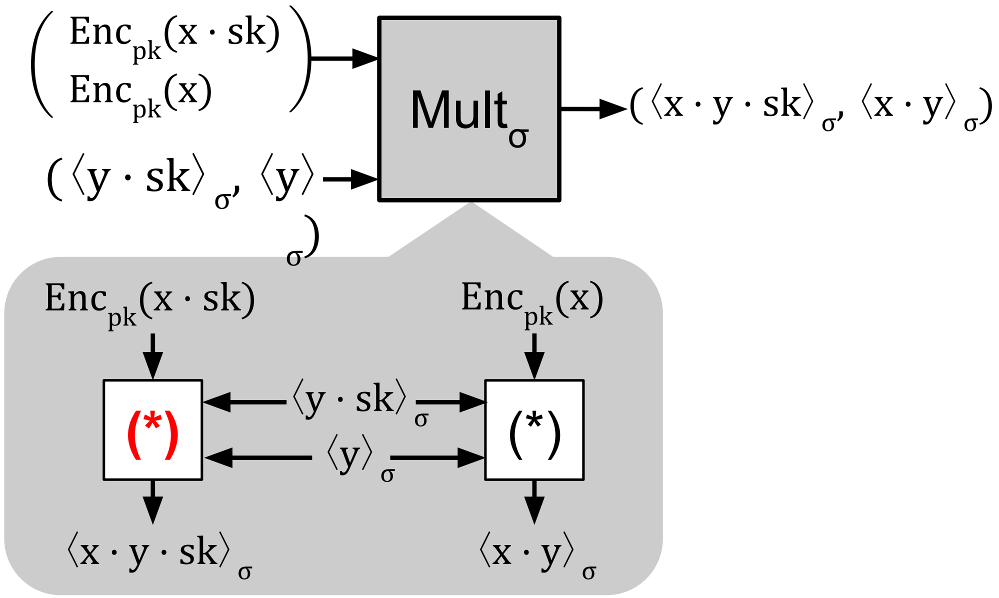
- start with *input encodings*
- intermediate computation values are computed as *memory shares*
- values held in memory shares can only be multiplied by values held in input encodings, not other values held in memory shares

input encoding of x: $\mathrm{Input\_Encode}(\mathrm{pk}, \mathrm{x}) = (\ \mathrm{Enc}_{\mathrm{pk}}(\mathrm{x} \cdot \mathrm{sk}),\ \mathrm{Enc}_{\mathrm{pk}}(\mathrm{x})\ )$

memory share of y: *subtractive shares* $(\langle \mathrm{y} \cdot \mathrm{sk} \rangle_\sigma, \langle \mathrm{y} \rangle_\sigma)$ $\mathrm{y} = \langle \mathrm{y} \rangle_A$ - $\langle \mathrm{y} \rangle_B$

need to be able to compute a memory share of xy given these

RMS multiplication: high level idea [Boyle-Gilboa-Ishai'16]



High level idea of (*):

1) Decrypt ciphertext
2) Multiply plaintext by y

in secret-shared form.

OSY'21 shows how to do this for DCR encodings

this computation requires **modular exponentiations**

Roadmap

1. Summary of our contributions
   a. Motivating example: two-party succinct secure computation
   b. Define multi-key homomorphic secret sharing (MKHSS)
   c. Application: non-interactive conditional key exchange
2. Background on HSS from DCR
3. Constructing MKHSS from the DCR assumption

next: Kevin will talk about optimized implementations of MKHSS/key exchange

Removing correlated setup [CDHJS’25]

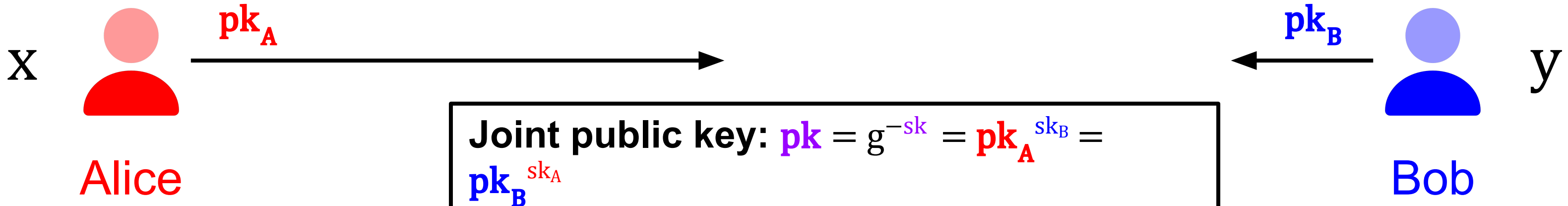
0. assume that Alice and Bob have pk and shares of sk
1. they exchange encodings of their inputs x and y under pk
2. they perform local computations to obtain shares of output f(x,y)

now we will see how to remove the assumption in 1 by having Alice and Bob

- use Diffie-Hellman key exchange to agree on a joint pk
- synchronize their input encodings under the joint pk
- (shares of joint sk are easy to generate with existing tools)

the rest of my part of the talk

Alice and Bob agree on joint key



Joint public key: $pk = g^{-sk} = pk_A{}^{sk_B} = pk_B{}^{sk_A}$

Joint secret key: $sk = sk_A \cdot sk_B$

Synchronizing Alice's input encoding

x Alice $\xrightarrow{\mathbf{pk_A}\quad \begin{pmatrix} \mathrm{Enc}_{\mathbf{pk_A}}(x \cdot sk_A) \\ \mathrm{Enc}_{\mathbf{pk_A}}(x) \end{pmatrix}}$

$\xleftarrow{\mathbf{pk_B}}$ Bob y

Joint public key: $\mathbf{pk} = g^{-sk} = \mathbf{pk_A}^{sk_B} = \mathbf{pk_B}^{sk_A}$
Joint secret key: $sk = sk_A \cdot sk_B$

① ②
both Alice and Bob need to compute Alice's synchronized input encoding:

$$\begin{pmatrix} \mathrm{Enc}_{\mathbf{pk}}(x \cdot sk) \\ \mathrm{Enc}_{\mathbf{pk}}(x) \end{pmatrix}$$

(Bob's encoding is synchronized symmetrically)

①: Alice syncs her own share

x Alice $\mathbf{pk_A}$ $\begin{pmatrix} \mathrm{Enc}_{\mathbf{pk_A}}(x \cdot sk_A) \\ \mathrm{Enc}_{\mathbf{pk_A}}(x) \end{pmatrix}$

$\mathbf{pk_B}$ Bob y

Joint public key: $\mathbf{pk} = g^{-sk} = \mathbf{pk_A}^{sk_B} = \mathbf{pk_B}^{sk_A}$
Joint secret key: $sk = sk_A \cdot sk_B$

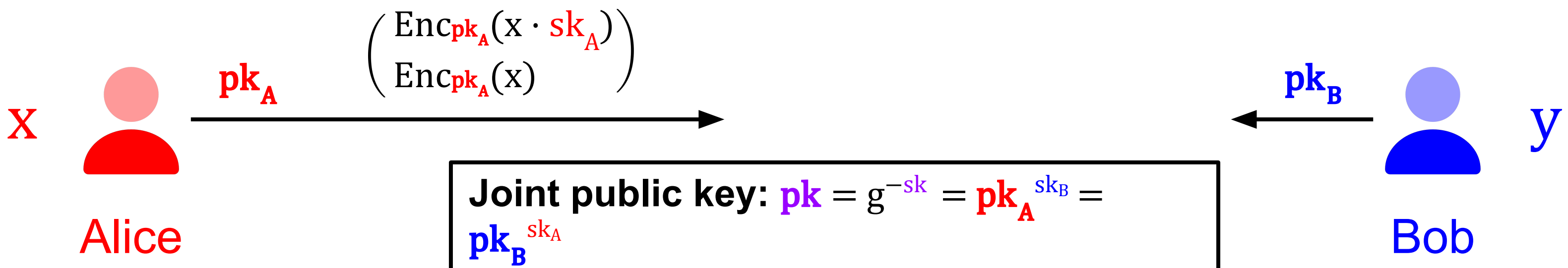
Flipped encryption:
Can compute **without** knowing secret key.

Re-encrypt under $\mathbf{pk}$

Re-encrypt x under $\mathbf{pk}$

$\begin{pmatrix} \mathrm{Enc}_{\mathbf{pk}}(x \cdot sk) \\ \mathrm{Enc}_{\mathbf{pk}}(x) \end{pmatrix}$

②: Bob syncs Alice's share



Joint public key: $\mathbf{pk} = g^{-sk} = \mathbf{pk_A}^{sk_B} = \mathbf{pk_B}^{sk_A}$
Joint secret key: $sk = sk_A \cdot sk_B$

$$\begin{pmatrix} \mathrm{Enc}_{\mathbf{pk_A}}(x \cdot sk_A) \\ \mathrm{Enc}_{\mathbf{pk_A}}(x) \end{pmatrix} \xrightarrow[\text{Multiply message/key by } sk_B]{\text{Multiply message/key by } sk_B} \begin{pmatrix} \mathrm{Enc}_{\mathbf{pk}}(x \cdot sk) \\ \mathrm{Enc}_{\mathbf{pk}}(x \cdot sk_B) \end{pmatrix}$$

Problem: junk term sk_B

plaintext space = $[N]$

solve by $sk_B = 1 \bmod N!$

but…

Issue with circular encryptions

problem: if sk is always $1 \bmod \mathrm{N}$, then we can no longer encode $\mathrm{x} \cdot sk$ in a plaintext space of $[\mathrm{N}]$

- recall that plaintexts are encoded in the exponent of $(1{+}\mathrm{N})$, which has order N

solution: compute circular encryptions $\bmod\ \mathrm{N}^4$ instead of $\bmod\ \mathrm{N}^2$! (i.e., generalized Damgard-Jurik encryptions)

in $\mathrm{Z}^*_{\mathrm{N}^4}$, the element $(1{+}\mathrm{N})$ has order N^3, so we have a plaintext space large enough to encode $\mathrm{x} \cdot sk$

Alice's input encoding

x Alice $\xrightarrow{pk_A} \begin{pmatrix} Enc_{pk_A}(x \cdot sk_A) & (\mathrm{mod}\ N^4) \\ Enc_{pk_A}(x) & (\mathrm{mod}\ N^2) \end{pmatrix}$

$\xleftarrow{pk_B}$ Bob y

Joint public key: $pk = g^{-sk} = pk_A{}^{sk_B} = pk_B{}^{sk_A}$
Joint secret key: $sk = sk_A \cdot sk_B$

✓ ✓

both Alice and Bob need to compute Alice's synchronized input encoding:

$$\begin{pmatrix} Enc_{pk}(x \cdot sk) & (\mathrm{mod}\ N^4) \\ Enc_{pk}(x) & (\mathrm{mod}\ N^2) \end{pmatrix}$$

Why is sampling the secret key this way secure?

- instead of sampling $\mathrm{sk} \leftarrow\$ [N]$, we now sample $\mathrm{sk'} \leftarrow\$ \{0,\ldots,N\text{-}1\}$ and set $\mathrm{sk} = \mathrm{sk'} \cdot N + 1$ so that $\mathrm{sk} = 1 \bmod N$

- note that g has order $\phi(N)/4$, which is coprime to N

- so the distribution over public keys $\mathrm{pk} = g^{-\mathrm{sk}}$ remains statistically close to the old distribution over public keys

Aside: Short Exponent Assumption

essentially says: sampling much shorter sk is still secure

- for this construction, no need to make this assumption to prove security
  - construction for class groups does require making this assumption
- but it allows sampling *much smaller keys* in practice
  - no longer have statistical closeness to original distribution of Pailler ElGamal public keys

Roadmap

1. Summary of our contributions
   a. Motivating example: two-party succinct secure computation
   b. Define multi-key homomorphic secret sharing (MKHSS)
   c. Application: non-interactive conditional key exchange
2. Background on HSS from DCR
3. Constructing MKHSS from the DCR assumption

next: Kevin will talk about optimized implementations of MKHSS/key exchange

Concretely-Efficient Multi-Key Homomorphic Secret Sharing and Applications

Kaiwen (Kevin) He, Sacha Servan-Schreiber, Geoffroy Couteau, Srini Devadas

To appear at *IEEE S&P* 2026

Roadmap

1. Overview of our work
2. MKHSS optimizations
3. Non-interactive conditional key exchange optimizations
4. Useful instantiations of key exchange
    a. Fuzzy password-authenticated key exchange
    b. Geolocation-based key exchange
5. Performance evaluation
6. Future works and conclusion

Roadmap

1. Overview of our work
2. MKHSS optimizations
3. Non-interactive conditional key exchange optimizations
4. Useful instantiations of key exchange
    a. Fuzzy password-authenticated key exchange
    b. Geolocation-based key exchange
5. Performance evaluation
6. Future works and conclusion

Recall: Multi-Key HSS Syntax [CDHJS'25]

Function: f

x Alice $\xrightarrow{pe_A^{(x)}}$ $\xleftarrow{pe_B^{(y)}}$ Bob y

Alice $\downarrow$ k_A

Bob $\downarrow$ k_B

$$k_A - k_B = f(x,y)$$

$pe_A^{(x)}, st_A \leftarrow \mathrm{Encode}(x)$

$k_A \leftarrow \mathrm{Eval}_A(f, pe_B^{(y)}, st_A)$

$\mathrm{Encode}(y) \rightarrow pe_B^{(y)}, st_B$

$\mathrm{Eval}_B(f, pe_A^{(x)}, st_B) \rightarrow k_B$

Application: Non-Interactive Conditional Key Exchange [CDHJS'25]

$$\boxed{x \approx y \Leftrightarrow f(x, y) = 1}$$

x Alice $\xrightarrow{pe_A^{(x)}}$ $\xleftarrow{pe_B^{(y)}}$ Bob y

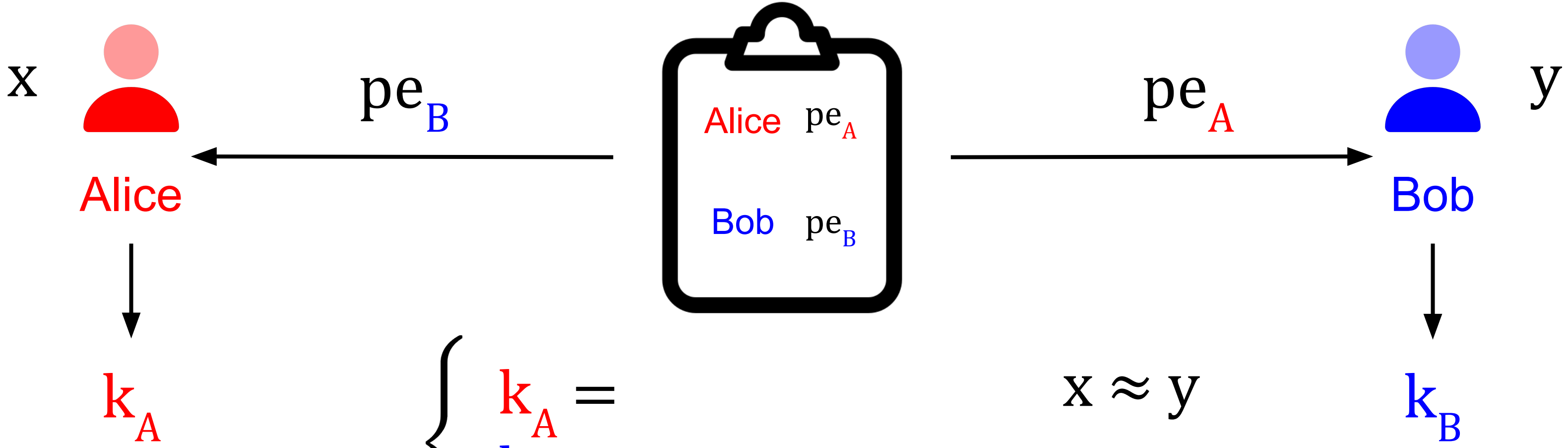
$$\begin{cases} k_A = k_B & x \approx y \\ \text{Independent } k_A \text{ and } k_B & x \not\approx y \end{cases}$$

Alice $\downarrow$ k_A

Bob $\downarrow$ k_B

Application: Non-Interactive Conditional Key Exchange [CDHJS'25]

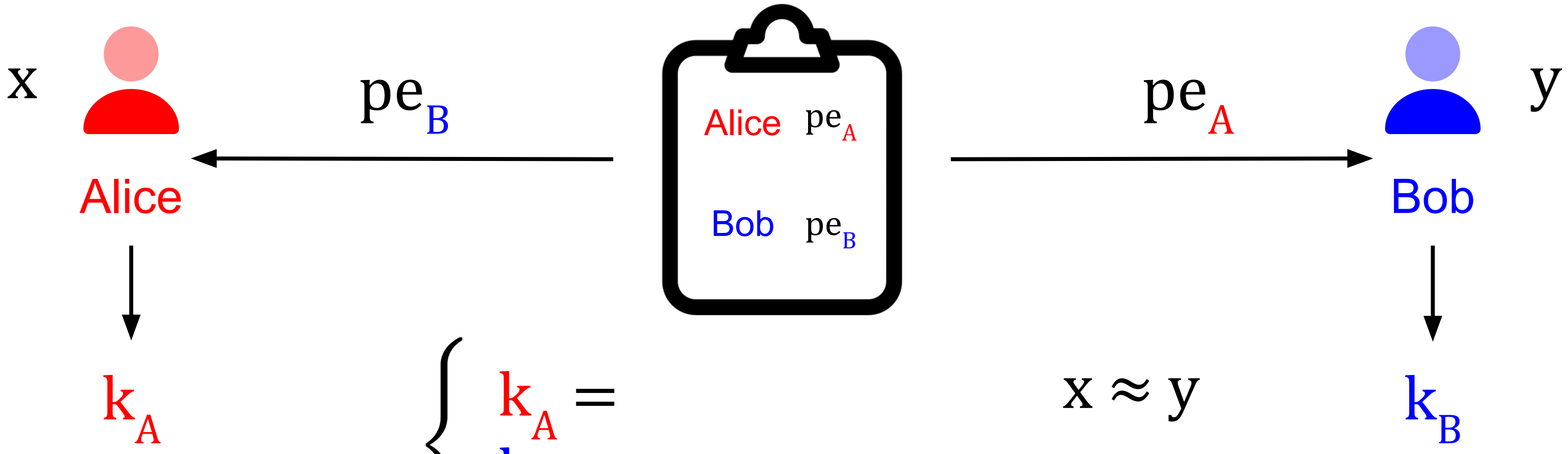
$$\boxed{x \approx y \Leftrightarrow f(x, y) = 1}$$



$$\begin{cases} k_A = k_B & x \approx y \\ \text{Independent } k_A \text{ and } k_B & x \not\approx y \end{cases}$$

Application: Non-Interactive Conditional Key Exchange [CDHJS'25]

$$x \approx y \Leftrightarrow f(x, y) = 1$$



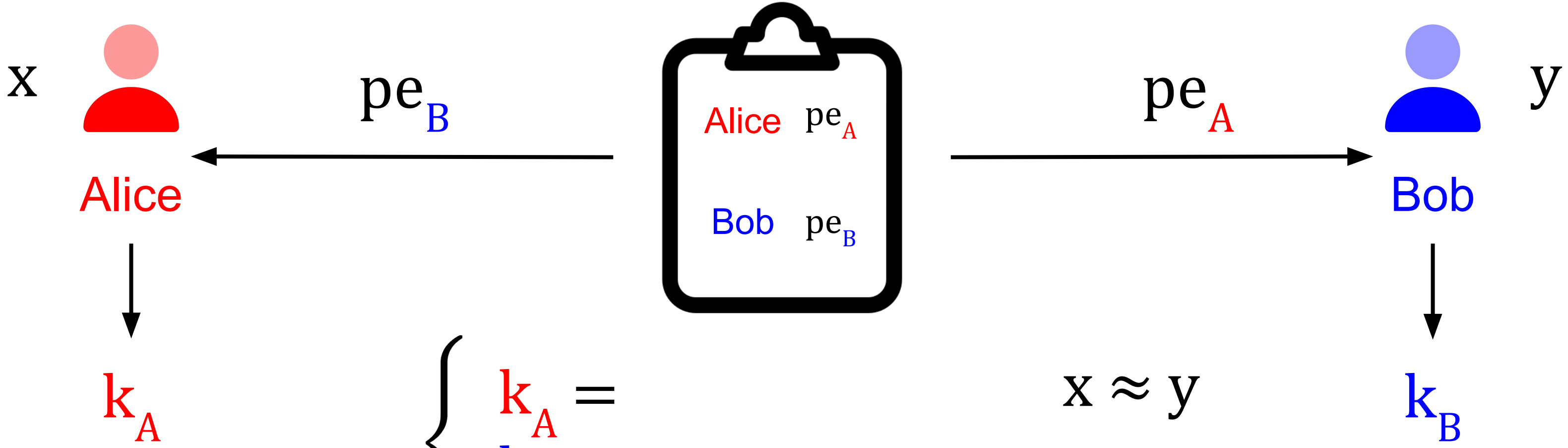
$$\begin{cases} k_A = k_B & x \approx y \\ \text{Independent } k_A \text{ and } k_B & x \not\approx y \end{cases}$$

⭐ A natural generalization of Diffie-Hellman-style key exchange [DH'76, FHKP'13]

Application: Non-Interactive Conditional Key Exchange [CDHJS'25]

$$x \approx y \Leftrightarrow \mathbf{f}(x, y) = 1$$

Concrete instantiations of the predicate $\mathbf{f}$?



$$\begin{cases} k_A = k_B & x \approx y \\ \text{Independent } k_A \text{ and } k_B & x \not\approx y \end{cases}$$

⭐ A natural generalization of Diffie-Hellman-style key exchange [DH'76, FHKP'13]

Concrete Instantiation: fPAKE [DHPRY'18]

$$x \approx y \Leftrightarrow \mathrm{EditDistance}(x, y) \leq T$$

fPAKE: **fuzzy** password-authenticated key exchange

correct
horse
battery
staple
$= x$

$y =$
corrupt
hose
buttery
stable

Alice	pe_A
Bob	pe_B

Alice $\xleftarrow{pe_B}$ | $\xrightarrow{pe_A}$ Bob

Alice $\rightarrow k_A$

Bob $\rightarrow k_B$

$$\begin{cases} k_A = k_B & x \approx y \\ \text{Independent } k_A \text{ and } k_B & x \not\approx y \end{cases}$$

Prior work requires 5+ rounds of interaction

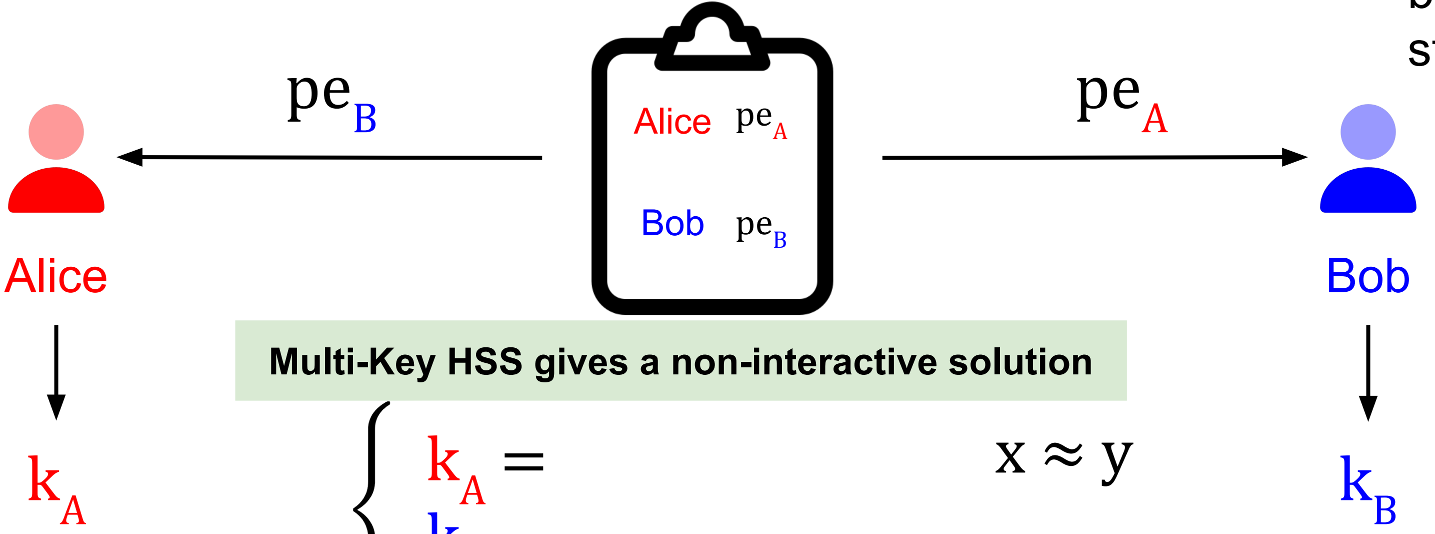
Concrete Instantiation: fPAKE [DHPRY'18]

$$x \approx y \Leftrightarrow \mathrm{EditDistance}(x, y) \leq T$$

correct horse battery staple $= x$

fPAKE: **fuzzy** password-authenticated key exchange

$y =$ corrupt hose buttery stable

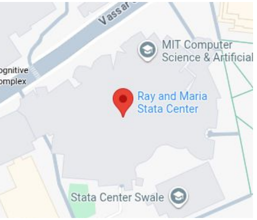


Multi-Key HSS gives a non-interactive solution

$$\begin{cases} k_A = k_B & x \approx y \\ \text{Independent } k_A \text{ and } k_B & x \not\approx y \end{cases}$$

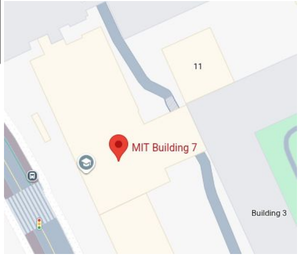
Concrete Instantiation: Geolocation-Based Key Exchange

$$\boxed{x \approx y \Leftrightarrow \text{Distance}(x, y) \leq T}$$



$= x$

$y =$



pe_B ← Alice | Alice pe_A, Bob pe_B | → pe_A Bob

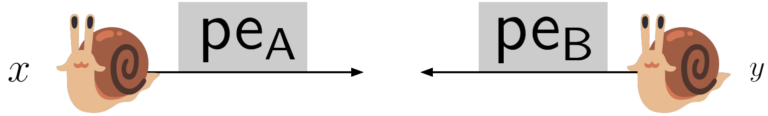
Alice → k_A

Bob → k_B

$$\begin{cases} k_A = k_B & x \approx y \\ \text{Independent } k_A \text{ and } k_B & x \not\approx y \end{cases}$$

Our Contributions

Prior work **[CDHJS'25]**



❌ Theoretical feasibility result, no code

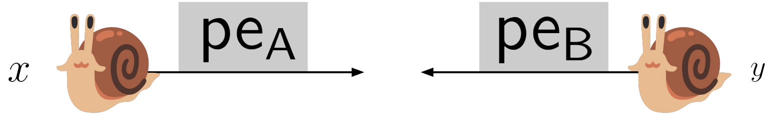
Our work

$$x \xrightarrow{\mathsf{pe}_A} \qquad \xleftarrow{\mathsf{pe}_B} y$$

✅ Open-source implementation

Our Contributions

Prior work [CDHJS'25]



❌ Theoretical feasibility result, no code

❌ A multiplication takes **224.6 ms** (if implemented)

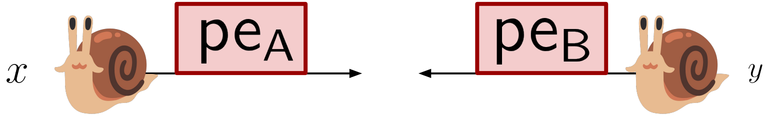
Our work

✅ Open-source implementation

✅ A multiplication takes **5.0 ms** (**45×** speedup)

Our Contributions

Prior work [CDHJS'25]



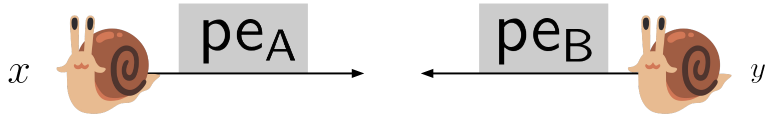
- ❌ Theoretical feasibility result, no code
- ❌ A multiplication takes **224.6 ms** (if implemented)
- ❌ Large communication overhead

Our work

- ✅ Open-source implementation
- ✅ A multiplication takes **5.0 ms** (**45×** speedup)
- ✅ **3×** reduction in communication for all apps

Our Contributions

Prior work **[CDHJS’25]**



❌ Theoretical feasibility result, no code

❌ A multiplication takes **224.6 ms** (if implemented)

❌ Large communication overhead

❌ Did not develop concrete applications

- Mentioned fPAKE in passing without giving a concrete instantiation

Our work

✅ Open-source implementation

✅ A multiplication takes **5.0 ms** (**45×** speedup)

✅ **3×** reduction in communication for all apps

✅ Identifies two useful applications of MKHSS:

- fPAKE
- Geolocation-based key exchange

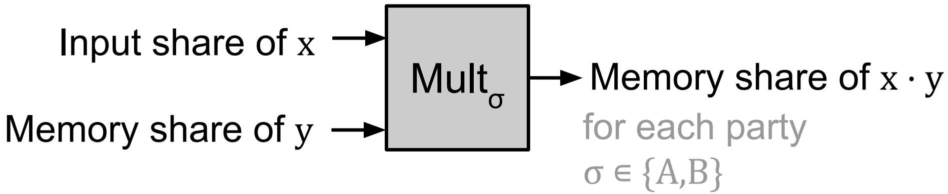
In addition, each app runs in a few seconds.

Roadmap

1. Overview of our work
2. MKHSS optimizations
3. Non-interactive conditional key exchange optimizations
4. Useful instantiations of key exchange
    a. Fuzzy PAKE
    b. Geolocation-based key exchange
5. Performance evaluation
6. Future works and conclusion

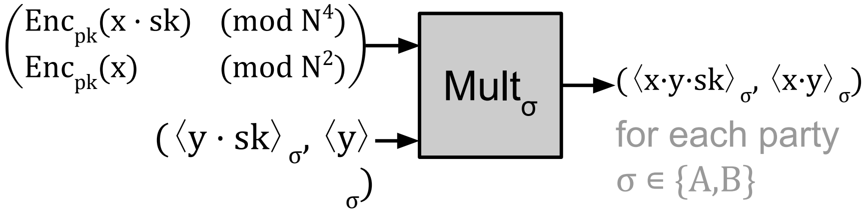
Bottleneck of (MK)HSS: RMS Multiplication

RMS Multiplication



Bottleneck of (MK)HSS: RMS Multiplication

Prior work [CDHJS'25]



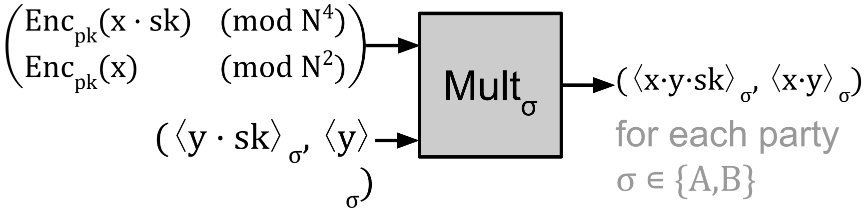
Notation

$$x = \langle x \rangle_A - \langle x \rangle_B$$

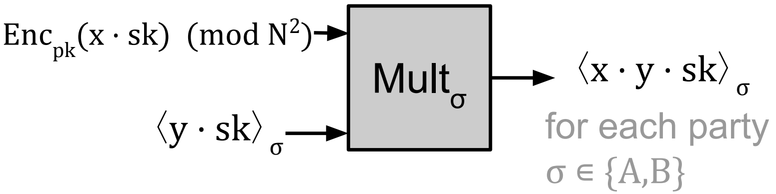
$$\mathrm{Enc}_{pk}(x) = (g^r, pk^r \cdot (1+N)^x)$$ [BCP'03, DJ'03]

Overview Of Our Optimizations

Prior work [CDHJS’25]



Our work



Notation

$$x = \langle x \rangle_A - \langle x \rangle_B$$

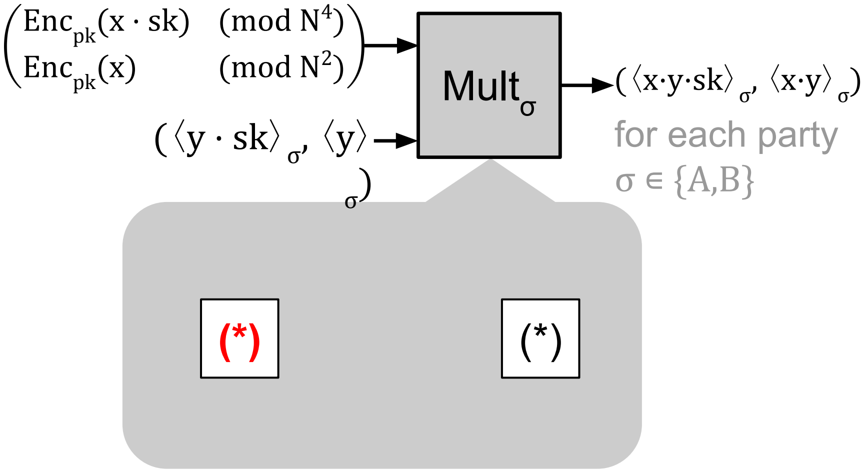
$$\mathrm{Enc}_{pk}(x) = (g^r, pk^r \cdot (1+N)^x)$$ [BCP’03, DJ’03]

Overview Of Our Optimizations

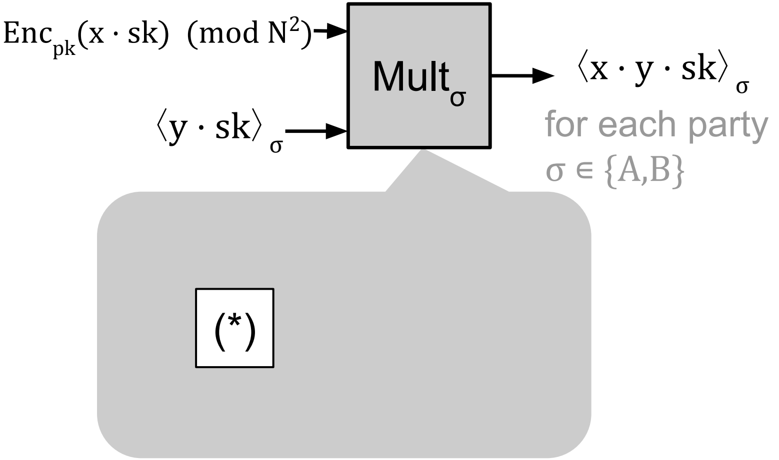
Notation

$x = \langle x \rangle_A - \langle x \rangle_B$

Prior work [CDHJS'25]



Our work



(*) : two exponentiations mod $\mathbf{N^4}$

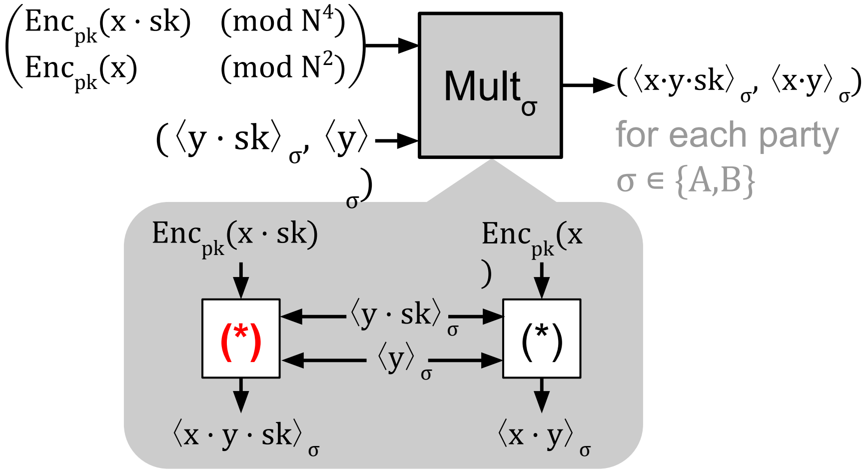
(*) : two exponentiations mod N^2

Overview Of Our Optimizations

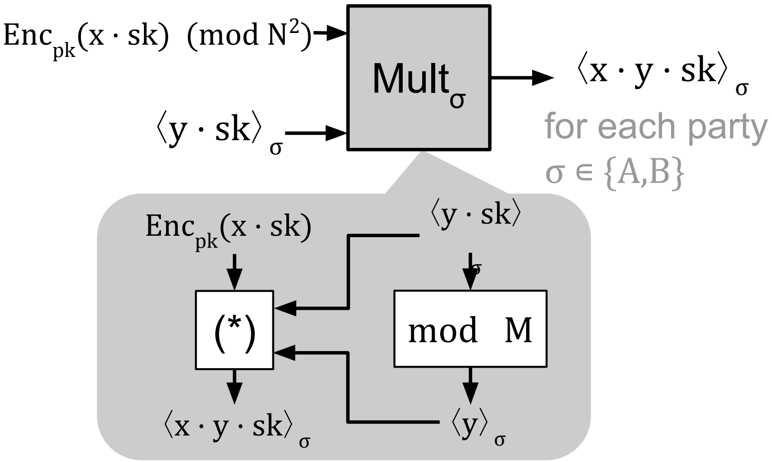
Notation

$x = \langle x\rangle_A - \langle x\rangle_B$

Prior work [CDHJS'25]



Our work



A key procedure used by much of the HSS literature [BGI'16]

(*) : two exponentiations mod N^4

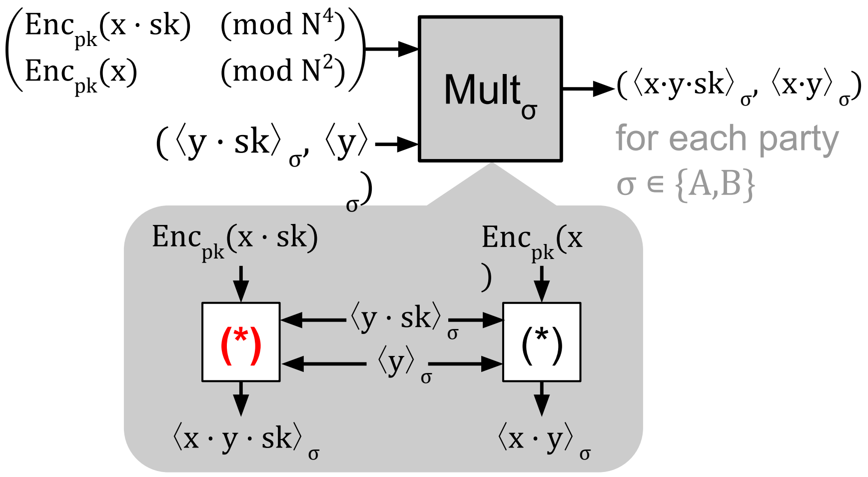
(*) : two exponentiations mod N^2

Key Procedure of Our Work

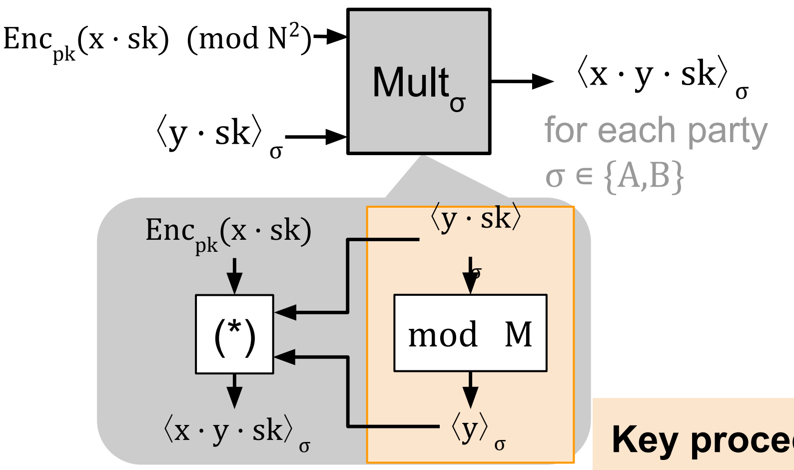
Notation

$x = \langle x\rangle_A - \langle x\rangle_B$

Prior work [CDHJS'25]



Our work



Key procedure of our work

(*) : two exponentiations mod $\mathbf{N^4}$

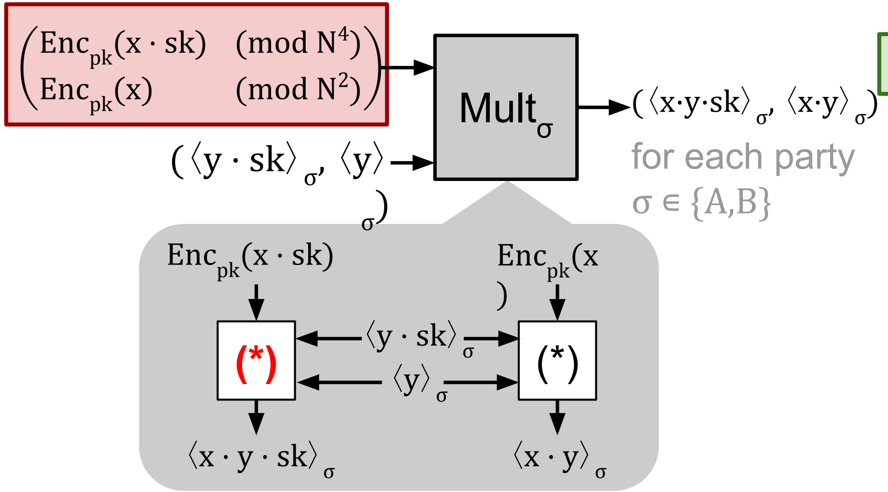
(*) : two exponentiations mod N^2

Simplifying Input Shares

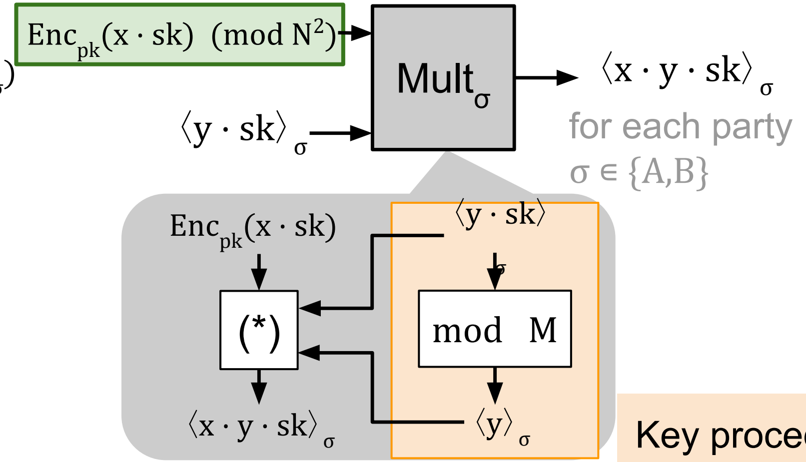
Notation

$x = \langle x \rangle_A - \langle x \rangle_B$

Input Share **Prior work** [CDHJS'25]



Input Share **Our work**



Key procedure of our work

Crucially simplifies **input share** structure.

$\boxed{(*)}$: two exponentiations mod $\mathbf{N^4}$

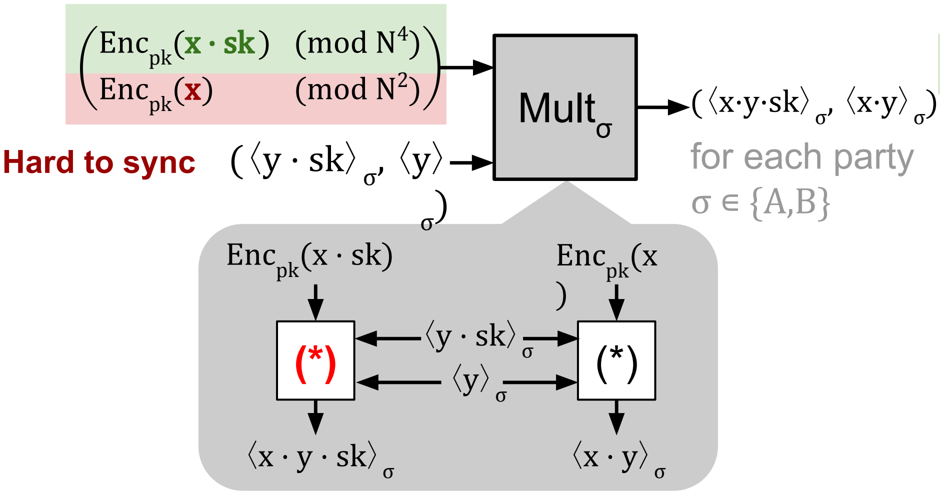
$\boxed{(*)}$: two exponentiations mod N^2

Making Share Synchronization Easier

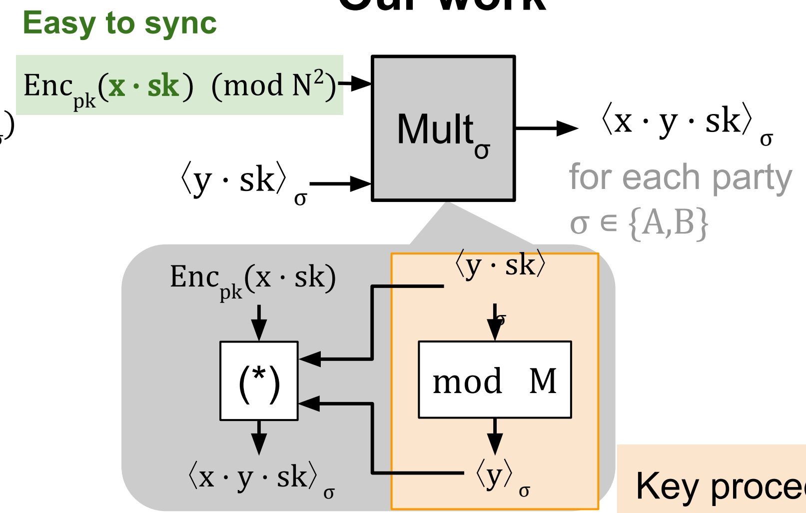
Notation

$x = \langle x \rangle_A - \langle x \rangle_B$

Prior work [CDHJS'25]



Our work



Key procedure of our work

Crucially simplifies **input share structure**.

Share synchronization [CDHJS'25] is a key step to realize multi-key HSS.

$\boxed{(*)}$: two exponentiations mod $\mathbf{N^4}$

$\boxed{(*)}$: two exponentiations mod N^2

Making Share Synchronization Easier

Notation

$x = \langle x \rangle_A - \langle x \rangle_B$

Prior work [CDHJS'25]

Our work

Share synchronization [CDHJS'25] is a key step to realize multi-key HSS.

Crucially simplifies **input share structure**.

$\boxed{(*)}$: two exponentiations mod N^4

$\boxed{(*)}$: two exponentiations mod N^2

Review: Share Synchronization [CDHJS'25]

Input share before: $\begin{pmatrix} \mathrm{Enc}_{\mathrm{pk}}(\mathrm{x} \cdot \mathrm{sk}) & (\mathrm{mod}\ \mathrm{N}^4) \\ \mathrm{Enc}_{\mathrm{pk}}(\mathrm{x}) & (\mathrm{mod}\ \mathrm{N}^2) \end{pmatrix}$

Cheatsheet

Secret keys of Alice and Bob: sk_A, sk_B

Joint secret key: $\mathrm{sk} = \mathrm{sk}_\mathrm{A} \cdot \mathrm{sk}_\mathrm{B}$

Public key of party σ: $\mathrm{pk}_\sigma \equiv \mathrm{g}^{-\mathrm{sk}_\sigma} \quad (\mathrm{mod}\ \mathrm{N}^{\mathrm{w}+1})$

Joint public key: $\mathrm{pk} \equiv \mathrm{g}^{-\mathrm{sk}} \equiv {\mathrm{pk}_\mathrm{A}}^{\mathrm{sk}_\mathrm{B}} \equiv {\mathrm{pk}_\mathrm{B}}^{\mathrm{sk}_\mathrm{A}}$ $(\mathrm{mod}\ \mathrm{N}^{\mathrm{w}+1})$

Review: Share Synchronization [CDHJS'25]

Input share before: $\begin{pmatrix} \mathrm{Enc}_{pk}(x \cdot sk) & \pmod{N^4} \\ \mathrm{Enc}_{pk}(x) & \pmod{N^2} \end{pmatrix}$

Easy to synchronize $\mathrm{Enc}_{pk}(x \cdot sk)$:

$\mathrm{Enc}_{\mathbf{pk}}(x \cdot sk) = \mathrm{Mul}(\mathrm{Enc}_{\mathbf{pk_A}}(x \cdot sk_A), sk_B)$

Cheatsheet

Secret keys of Alice and Bob: sk_A, sk_B

Joint secret key: $sk = sk_A \cdot sk_B$

Public key of party σ: $\mathbf{pk_\sigma} \equiv g^{-sk_\sigma} \pmod{N^{w+1}}$

Joint public key: $\mathbf{pk} \equiv g^{-sk} \equiv \mathbf{pk_A}^{sk_B} \equiv \mathbf{pk_B}^{sk_A} \pmod{N^{w+1}}$

$\mathrm{Mul}(ct, u)$ multiplies both key and message by u.

Review: Share Synchronization [CDHJS'25]

Input share before: $\begin{pmatrix} \mathrm{Enc}_{pk}(x \cdot sk) & \pmod{N^4} \\ \mathrm{Enc}_{pk}(x) & \pmod{N^2} \end{pmatrix}$

Easy to synchronize $\mathrm{Enc}_{pk}(x \cdot sk)$:

$\mathrm{Enc}_{\mathbf{pk}}(x \cdot sk) = \mathrm{Mul}(\mathrm{Enc}_{\mathbf{pk_A}}(x \cdot sk_A), sk_B)$

Hard to synchronize $\mathrm{Enc}_{pk}(x)$:

$\mathrm{Enc}_{\mathbf{pk}}(x) = \mathrm{Enc}_{\mathbf{pk}}(x \cdot sk_B) = \mathrm{Mul}(\mathrm{Enc}_{\mathbf{pk_A}}(x), sk_B)$

Requires $sk_B = sk_B' \cdot N + 1$ (Likewise for sk_A)

Cheatsheet

Secret keys of Alice and Bob: sk_A, sk_B

Joint secret key: $sk = sk_A \cdot sk_B$

Public key of party σ: $\mathbf{pk_\sigma} \equiv g^{-sk_\sigma} \pmod{N^{w+1}}$

Joint public key: $\mathbf{pk} \equiv g^{-sk} \equiv \mathbf{pk_A}^{sk_B} \equiv \mathbf{pk_B}^{sk_A} \pmod{N^{w+1}}$

$\mathrm{Mul}(ct, u)$ multiplies both key and message by u.

Review: Share Synchronization [CDHJS'25]

Input share before: $\begin{pmatrix} \mathrm{Enc}_{pk}(x \cdot sk) & \pmod{N^4} \\ \mathrm{Enc}_{pk}(x) & \pmod{N^2} \end{pmatrix}$

Easy to synchronize $\mathrm{Enc}_{pk}(x \cdot sk)$:

$\mathrm{Enc}_{\mathbf{pk}}(x \cdot sk) = \mathrm{Mul}(\mathrm{Enc}_{\mathbf{pk_A}}(x \cdot sk_A), sk_B)$

Hard to synchronize $\mathrm{Enc}_{pk}(x)$:

$\mathrm{Enc}_{\mathbf{pk}}(x) = \mathrm{Enc}_{\mathbf{pk}}(x \cdot sk_B) = \mathrm{Mul}(\mathrm{Enc}_{\mathbf{pk_A}}(x), sk_B)$

Requires $sk_B = sk_B' \cdot N + 1$ (Likewise for sk_A)

Problem: large joint secret key:

$|x \cdot sk| \approx N^2 \cdot |x| \cdot |sk_A'| \cdot |sk_B'| \gg N^2$

Cheatsheet

Secret keys of Alice and Bob: sk_A, sk_B

Joint secret key: $sk = sk_A \cdot sk_B$

Public key of party σ: $\mathbf{pk_\sigma} \equiv g^{-sk_\sigma} \pmod{N^{w+1}}$

Joint public key: $\mathbf{pk} \equiv g^{-sk} \equiv \mathbf{pk_A}^{sk_B} \equiv \mathbf{pk_B}^{sk_A} \pmod{N^{w+1}}$

$\mathrm{Mul}(ct, u)$ multiplies both key and message by u.

Review: Share Synchronization [CDHJS'25]

Input share before: $\begin{pmatrix} \mathrm{Enc}_{pk}(x \cdot sk) & \pmod{N^4} \\ \mathrm{Enc}_{pk}(x) & \pmod{N^2} \end{pmatrix}$

Easy to synchronize $\mathrm{Enc}_{pk}(x \cdot sk)$:

$\mathrm{Enc}_{\mathbf{pk}}(x \cdot sk) = \mathrm{Mul}(\mathrm{Enc}_{\mathbf{pk_A}}(x \cdot sk_A), sk_B)$

Hard to synchronize $\mathrm{Enc}_{pk}(x)$:

$\mathrm{Enc}_{\mathbf{pk}}(x) = \mathrm{Enc}_{\mathbf{pk}}(x \cdot sk_B) = \mathrm{Mul}(\mathrm{Enc}_{\mathbf{pk_A}}(x), sk_B)$

Requires $sk_B = sk_B' \cdot N + 1$ (Likewise for sk_A)

Problem: large joint secret key:

$|x \cdot sk| \approx N^2 \cdot |x| \cdot |sk_A'| \cdot |sk_B'| \gg N^2$

For $\mathrm{Enc}_{pk}(x \cdot sk)$ to decrypt correctly, need modulus N^4 *even when* we use short exponents: $|sk_\sigma'| \approx 2^{2\lambda}$.

Cheatsheet

Secret keys of Alice and Bob: sk_A, sk_B

Joint secret key: $sk = sk_A \cdot sk_B$

Public key of party σ: $\mathbf{pk_\sigma} \equiv g^{-sk_\sigma} \pmod{N^{w+1}}$

Joint public key: $\mathbf{pk} \equiv g^{-sk} \equiv \mathbf{pk_A}^{sk_B} \equiv \mathbf{pk_B}^{sk_A} \pmod{N^{w+1}}$

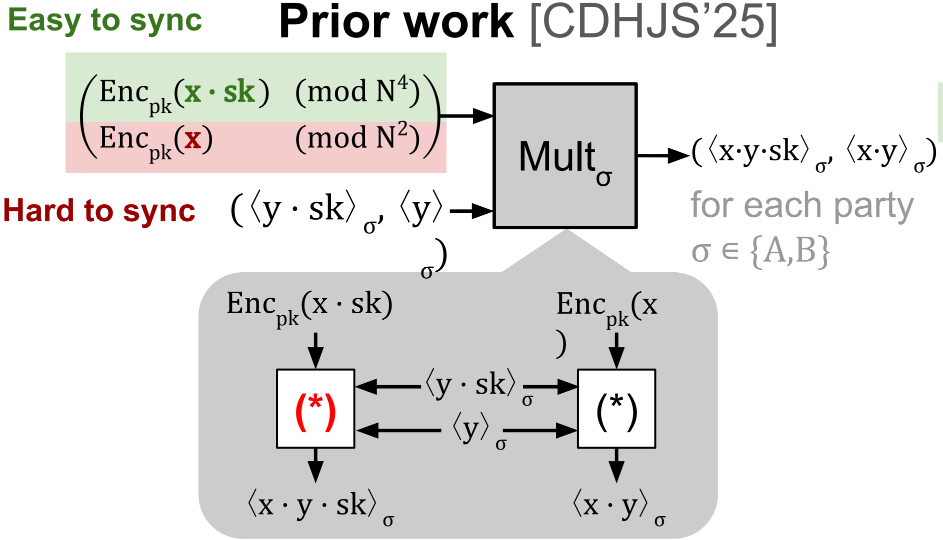
$\mathrm{Mul}(ct, u)$ multiplies both key and message by u.

Making Share Synchronization Easier

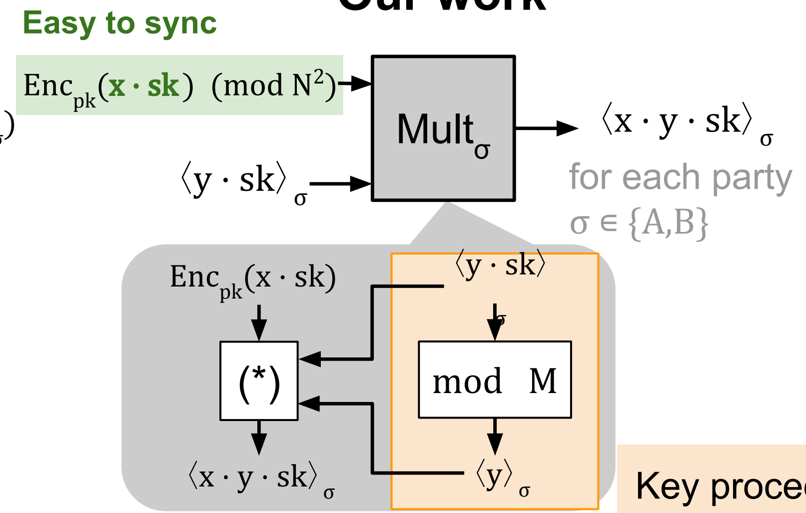
Notation

$x = \langle x \rangle_A - \langle x \rangle_B$

Prior work [CDHJS'25]



Our work



Key procedure of our work

✅ **We do not need arithmetic mod $\mathbf{N^4}$.**

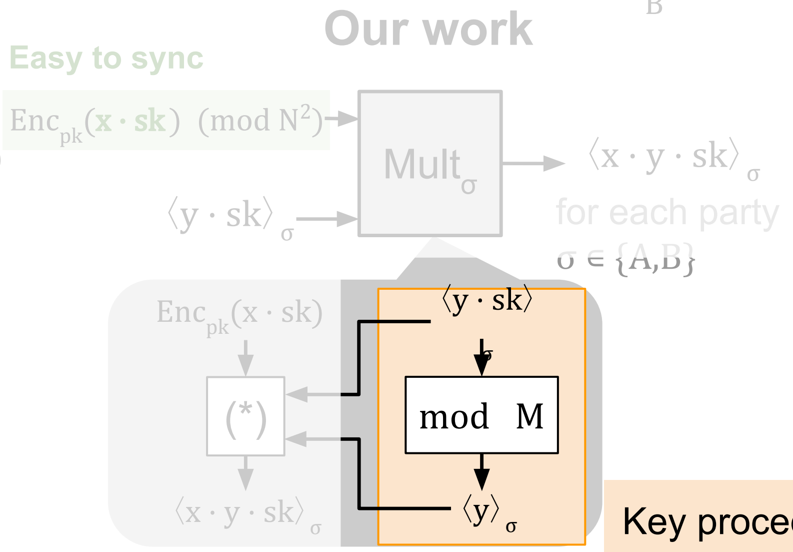
(*) : two exponentiations mod $\mathbf{N^4}$

(*) : two exponentiations mod N^2

Making Share Synchronization Easier

Notation

$x = \langle x \rangle_A - \langle x \rangle_B$



Key procedure of our work

✅ We do not need arithmetic mod N^4.

$\boxed{(*)}$: two exponentiations mod N^4

$\boxed{(*)}$: two exponentiations mod N^2

Simplifying Memory Shares **(This Work)**

Notation

$x = \langle x \rangle_A - \langle x \rangle_B$

Goal: derive $\langle y \rangle_\sigma$ locally from $\langle y \cdot sk \rangle_\sigma$

$\langle y \cdot sk \rangle_A$

$\langle y \cdot sk \rangle_B$

Simplifying Memory Shares **(This Work)**

Notation

$x = \langle x \rangle_A - \langle x \rangle_B$

Goal: derive $\langle y \rangle_\sigma$ locally from $\langle y \cdot sk \rangle_\sigma$

Precondition: Parties hold random shares

Achievable by applying a public random offset

$$r \leftarrow\$ \{0, \dots, N-1\}$$

$\langle y \cdot sk \rangle_A$

$= r + y \cdot sk$

$\langle y \cdot sk \rangle_B$

$= r$

Simplifying Memory Shares **(This Work)**

Notation

$x = {\color{red}\langle \mathbf{x} \rangle_A} - {\color{blue}\langle \mathbf{x} \rangle_B}$

Goal: derive $\langle y \rangle_\sigma$ locally from $\langle y \cdot sk \rangle_\sigma$

$${\color{red}\langle y \cdot sk \rangle_A} = r + y \cdot sk \xrightarrow{\bmod M} = \boxed{r'} + y \cdot sk \pmod M$$

$$r \leftarrow\$ \{0, \dots, N-1\}$$

$${\color{blue}\langle y \cdot sk \rangle_B} = r \xrightarrow{\bmod M} = \boxed{r'}$$

Define $r' := r \bmod M$

Simplifying Memory Shares **(This Work)**

Notation

$x = \langle x \rangle_A - \langle x \rangle_B$

Goal: derive $\langle y \rangle_\sigma$ locally from $\langle y \cdot sk \rangle_\sigma$

Pick $M \leq N \cdot 2^{-\lambda}$

$r \leftarrow\$ \{0, \dots, N-1\}$

$\langle y \cdot sk \rangle_A$

$= r + y \cdot sk$

$\big\downarrow$ mod M

$= \boxed{r'} + y \cdot sk \pmod{M}$

$\langle y \cdot sk \rangle_B$

$= r$

$\big\downarrow$ mod M

$= \boxed{r'}$

Observe: $r' \approx_s \{0, \dots, M-1\}$

Define $r' := r \mod M$

Simplifying Memory Shares **(This Work)**

Notation

$x = \langle x \rangle_A - \langle x \rangle_B$

Goal: derive $\langle y \rangle_\sigma$ locally from $\langle y \cdot sk \rangle_\sigma$

Pick $M \le N \cdot 2^{-\lambda}$

$r \leftarrow\$ \{0, \dots, N-1\}$

$$\langle y \cdot sk \rangle_A = r + y \cdot sk \xrightarrow{\bmod M} = r' + y \cdot sk \pmod{M}$$

$$\langle y \cdot sk \rangle_B = r \xrightarrow{\bmod M} = r'$$

Observe: $r' \approx_s \{0, \dots, M-1\}$

Define $r' := r \bmod M$

Simplifying Memory Shares **(This Work)**

Notation
$x = \langle x \rangle_A - \langle x \rangle_B$

Goal: derive $\langle y \rangle_\sigma$ locally from $\langle y \cdot sk \rangle_\sigma$

Pick $M \leq N \cdot 2^{-\lambda}$

$r \leftarrow\$ \{0, \dots, N-1\}$

$\langle y \cdot sk \rangle_A$

$= r + y \cdot sk$

$\big\downarrow \bmod M$

$= r' + y \cdot sk \pmod M$

$\langle y \cdot sk \rangle_B$

$= r$

$\big\downarrow \bmod M$

$= r'$

Idea: What if $sk \equiv 1 \pmod M$?

Observe: $r' \approx_s \{0, \dots, M-1\}$

Define $r' := r \bmod M$

Simplifying Memory Shares **(This Work)**

Notation

$x = \langle x \rangle_A - \langle x \rangle_B$

Goal: derive $\langle y \rangle_\sigma$ locally from $\langle y \cdot sk \rangle_\sigma$

Pick $M \leq N \cdot 2^{-\lambda}$

$r \leftarrow \$ \{0, \ldots, N-1\}$

$\langle y \cdot sk \rangle_A$

$= r + y \cdot sk$

$\Big\downarrow \bmod M$

$= r' + y \cdot sk \pmod M$

Sample sk_A, sk_B like in [CDHJS'25], except with M instead of N:

$sk'_\sigma \leftarrow \$ \{0, \ldots, 2^{2\lambda} - 1\}$

$sk_\sigma = sk'_\sigma \cdot M + 1$

$\langle y \cdot sk \rangle_B$

$= r$

$\Big\downarrow \bmod M$

$= r'$

$r' \approx_s \{0, \ldots, M-1\}$

$r' := r \bmod M$

Simplifying Memory Shares **(This Work)**

Notation
$x = \langle x \rangle_A - \langle x \rangle_B$

Goal: derive $\langle y \rangle_\sigma$ locally from $\langle y \cdot sk \rangle_\sigma$

$\boxed{\text{Pick } M \leq N \cdot 2^{-\lambda}}$

$r \leftarrow\$ \{0, \dots, N-1\}$

$\langle y \cdot sk \rangle_A$

$= r + y \cdot sk$

$\big\downarrow \bmod M$

$= r' + y \cdot \cancel{sk} \pmod M$

Sample sk_A, sk_B like in [CDHJS'25], except with M instead of N:

$sk'_\sigma \leftarrow\$ \{0, \dots, 2^{2\lambda} - 1\}$

$sk_\sigma = sk'_\sigma \cdot M + 1$

$\Longrightarrow sk := sk_A \cdot sk_B \equiv 1 \pmod M$

$\langle y \cdot sk \rangle_B$

$= r$

$\big\downarrow \bmod M$

$= r'$

$r' \approx_s \{0, \dots, M-1\}$

$r' := r \bmod M$

Simplifying Memory Shares **(This Work)**

Notation

$x = \langle x \rangle_A - \langle x \rangle_B$

Goal: derive $\langle y \rangle_\sigma$ locally from $\langle y \cdot sk \rangle_\sigma$

$\boxed{\text{Pick } M \leq N \cdot 2^{-\lambda}}$

$r \leftarrow\$ \{0, \ldots, N-1\}$

$$\langle y \cdot sk \rangle_A = r + y \cdot sk \xrightarrow{\bmod M} = r' + y \pmod M$$

$$\langle y \cdot sk \rangle_B = r \xrightarrow{\bmod M} = r'$$

Problem: subtractive shares must be over the integers

$r' \approx_s \{0, \ldots, M-1\}$

$r' := r \bmod M$

Simplifying Memory Shares **(This Work)**

Notation

$x = \langle x \rangle_A - \langle x \rangle_B$

Goal: derive $\langle y \rangle_\sigma$ locally from $\langle y \cdot sk \rangle_\sigma$

Pick $M \leq N \cdot 2^{-\lambda}$

$r \leftarrow\$ \{0, \dots, N-1\}$

$\langle y \cdot sk \rangle_A$

$= r + y \cdot sk$

$\downarrow$ mod M

$= r' + y \ (\mathrm{mod}\ M)$

$= r' + y \ \ (\text{over } \mathbb{Z}) = \langle y \rangle_A$

holds if $r' < M - y$

$\langle y \cdot sk \rangle_B$

$= r$

$\downarrow$ mod M

$= r' = \langle y \rangle_B$

$r' \approx_s \{0, \dots, M-1\}$

$r' := r \ \mathrm{mod}\ M$

Simplifying Memory Shares **(This Work)**

Notation

$x = \langle x \rangle_A - \langle x \rangle_B$

Goal: derive $\langle y \rangle_\sigma$ locally from $\langle y \cdot sk \rangle_\sigma$

Need $\mathbf{y \cdot 2^{\lambda} \leq M} \leq N \cdot 2^{-\lambda}$

$r \leftarrow\$ \{0, \dots, N-1\}$

$\langle y \cdot sk \rangle_A$

$= r + y \cdot sk$

$\downarrow \bmod M$

$= r' + y \pmod M$

$= r' + y \quad (\text{over } \mathbb{Z}) = \langle y \rangle_A$

holds if $r' < M - y$, which is true with probability $\mathbf{\geq 1 - 2^{-\lambda}}$

$r' \approx_s \{0, \dots, M-1\}$

$r' := r \bmod M$

$\langle y \cdot sk \rangle_B$

$= r$

$\downarrow \bmod M$

$= r' = \langle y \rangle_B$

Simplifying Memory Shares **(This Work)**

Notation

$x = \langle x \rangle_A - \langle x \rangle_B$

Goal: derive $\langle y \rangle_\sigma$ locally from $\langle y \cdot sk \rangle_\sigma$

$\boxed{\text{Pick } M = \max(y) \cdot 2^\lambda}$

Need $y \cdot 2^\lambda \leq M \leq N \cdot 2^{-\lambda}$

$r \leftarrow\$ \{0, \ldots, N-1\}$

$\langle y \cdot sk \rangle_A$

$= r + y \cdot sk$

$\Big\downarrow$ mod M

$= r' + y \pmod{M}$

$= r' + y \quad (\text{over } \mathbb{Z}) = \langle y \rangle_A$

holds if $r' < M - y$, which is true with probability $\geq 1 - 2^{-\lambda}$

$\langle y \cdot sk \rangle_B$

$= r$

$\Big\downarrow$ mod M

$= r' = \langle y \rangle_B$

$r' \approx_s \{0, \ldots, M-1\}$

$r' := r \mod M$

Simplifying Memory Shares **(This Work)**

Notation

$x = \langle x \rangle_A - \langle x \rangle_B$

Goal: derive $\langle y \rangle_\sigma$ locally from $\langle y \cdot sk \rangle_\sigma$

Pick $M = \mathbf{B} \cdot 2^\lambda$

Parameter:
$\mathbf{B}$: magnitude bound on memory shares
Looking ahead, our key exchange applications only need $\mathbf{B} = 1$

$$\langle y \cdot sk \rangle_A = r + y \cdot sk \xrightarrow{\bmod M} = r' + y \pmod{M} = r' + y \ (\text{over } \mathbb{Z}) = \langle y \rangle_A$$

$$\langle y \cdot sk \rangle_B = r \xrightarrow{\bmod M} = r' = \langle y \rangle_B$$

holds if $r' < M - y$, which is true with probability $\geq 1 - 2^{-\lambda}$

$$r' \approx_s \{0, \dots, M-1\}$$

$$r' := r \bmod M$$

Simplifying Memory Shares (This Work)

Notation

$x = \langle x \rangle_A - \langle x \rangle_B$

Goal: derive $\langle y \rangle_\sigma$ locally from $\langle y \cdot sk \rangle_\sigma$

Pick $M = \mathbf{B} \cdot 2^\lambda$

$\langle y \cdot sk \rangle_A$

$= r + y \cdot sk$

$\big\downarrow \bmod M$

$= r' + y \pmod{M}$

$= r' + y \quad (\text{over } \mathbb{Z}) = \langle y \rangle_A$

Secret key size analysis:

$sk'_\sigma \leftarrow\$ \{0, \dots, 2^{2\lambda} - 1\}$

$sk_\sigma = sk'_\sigma \cdot M + 1$

$\Longrightarrow sk := sk_A \cdot sk_B \equiv 1 \pmod{M}$

$sk \le \mathbf{B}^2 \cdot 2^{6\lambda}$

$\mathbf{B} = 1, \lambda = 128, N \approx 2^{3072} \Longrightarrow sk \le N$

$\langle y \cdot sk \rangle_B$

$= r$

$\big\downarrow \bmod M$

$= r' = \langle y \rangle_B$

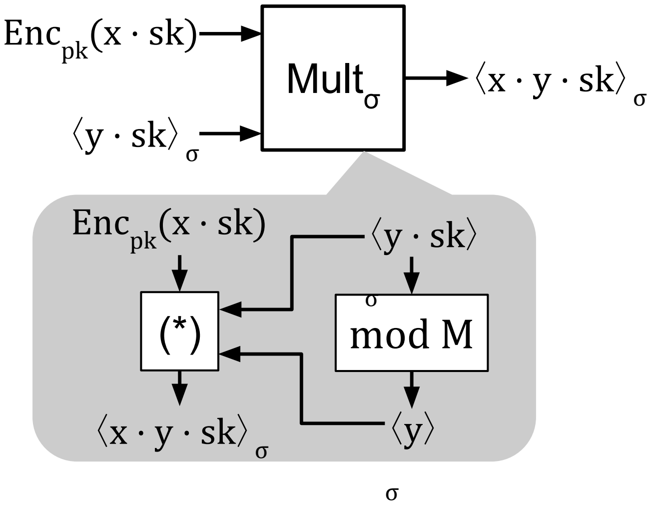
holds if $r' < M - y$, which is true with probability $\ge 1 - 2^{-\lambda}$

$r' \approx_s \{0, \dots, M - 1\}$

$r' := r \bmod M$

Our Optimization: Speed Up (*) By Shortening Exponent

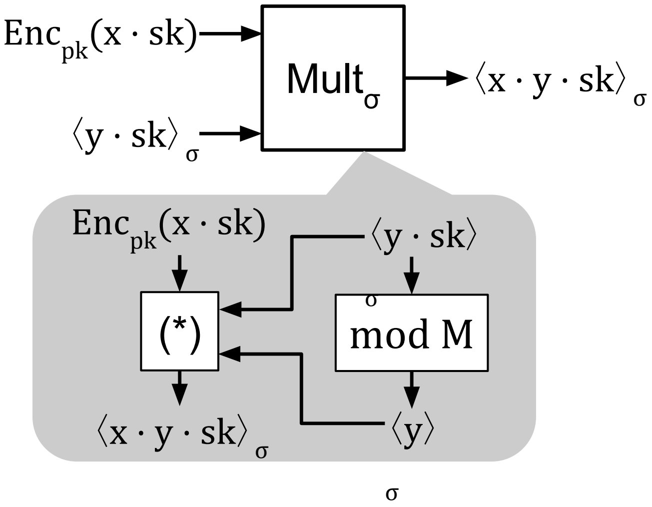
Recall our scheme



Running time of (*) ≈ two modular exponentiations

Our Optimization: Speed Up (*) By Shortening Exponent

Recall our scheme



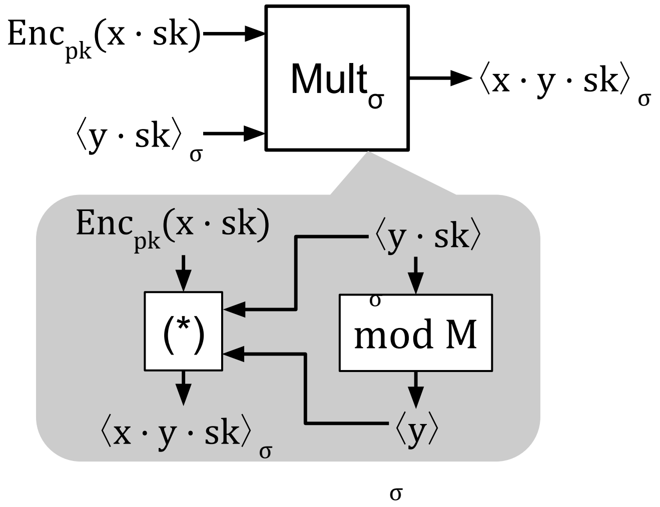
Running time of (*) ≈ two modular exponentiations
More precisely, this is the bottleneck:

$$ct_0^{\langle y \cdot sk \rangle_\sigma} \cdot ct_1^{\langle y \rangle_\sigma} \pmod{N^2}$$

where we unpack $\mathrm{Enc}_{pk}(x \cdot sk) = (ct_0, ct_1)$

Our Optimization: Speed Up (*) By Shortening Exponent

Recall our scheme



Running time of (*) ≈ two modular exponentiations
More precisely, this is the bottleneck:

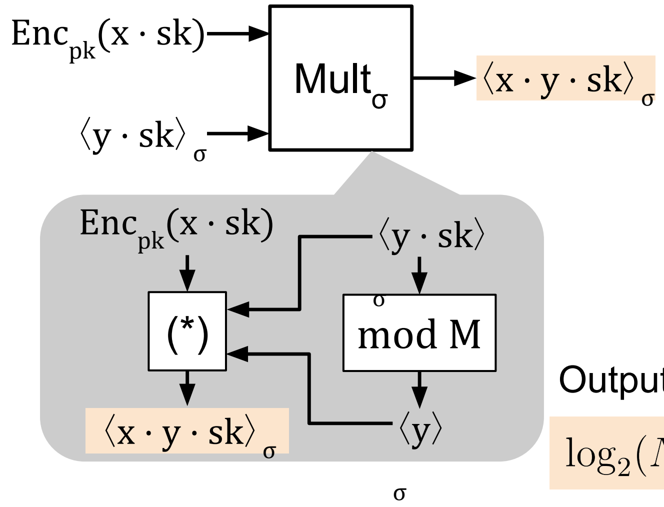
$$\mathrm{ct}_0^{\langle y \cdot sk \rangle_\sigma} \cdot \mathrm{ct}_1^{\langle y \rangle_\sigma} \pmod{N^2}$$

where we unpack $\mathrm{Enc}_{pk}(x \cdot sk) = (\mathrm{ct}_0, \mathrm{ct}_1)$

Time to compute modular exponentiation scales ~ **linearly** with the **length of the exponent**

Our Optimization: Speed Up (*) By Shortening Exponent

Recall our scheme



Running time of (*) ≈ two modular exponentiations
More precisely, this is the bottleneck:

$$ct_0^{\langle y \cdot sk \rangle_\sigma} \cdot ct_1^{\langle y \rangle_\sigma} \pmod{N^2}$$

where we unpack $\mathrm{Enc}_{pk}(x \cdot sk) = (ct_0, ct_1)$

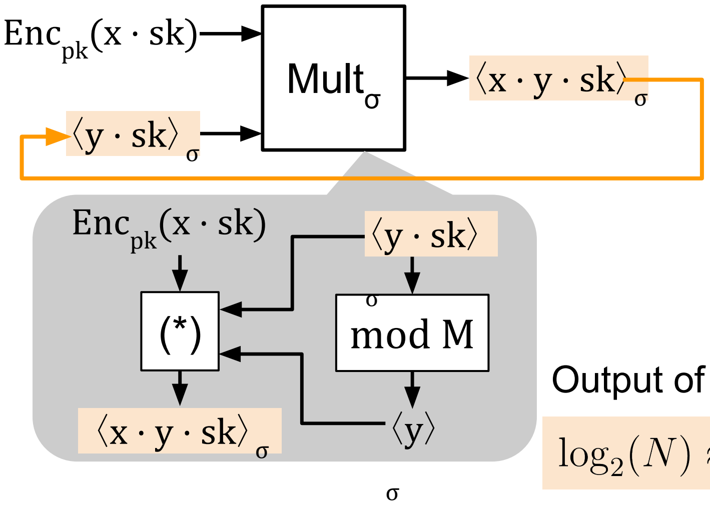
Output of (*) is:

$\log_2(N) \approx 3072$ bits

Time to compute modular exponentiation scales ~ **linearly** with the **length of the exponent**

Our Optimization: Speed Up (*) By Shortening Exponent

Recall our scheme



Running time of (*) ≈ two modular exponentiations
More precisely, this is the bottleneck:

$$ct_0^{\langle y \cdot sk \rangle_\sigma} \cdot ct_1^{\langle y \rangle_\sigma} \pmod{N^2}$$

where we unpack $\mathrm{Enc}_{pk}(x \cdot sk) = (ct_0, ct_1)$

3072 bits

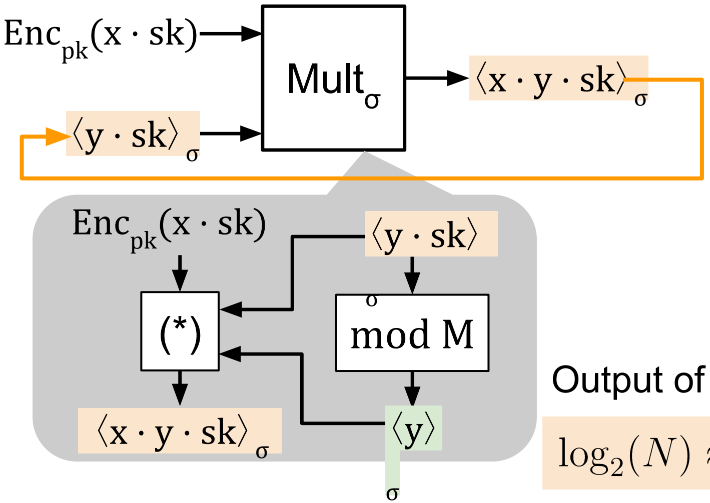
Output of (*) is:

$\log_2(N) \approx 3072$ bits

Time to compute modular exponentiation scales ~ **linearly** with the **length of the exponent**

Our Optimization: Speed Up (*) By Shortening Exponent

Recall our scheme



Running time of (*) ≈ two modular exponentiations
More precisely, this is the bottleneck:

$$\mathrm{ct}_0^{\langle y \cdot sk \rangle_\sigma} \cdot \mathrm{ct}_1^{\langle y \rangle_\sigma} \pmod{N^2}$$

where we unpack $\mathrm{Enc}_{pk}(x \cdot sk) = (\mathrm{ct}_0, \mathrm{ct}_1)$

3072 bits

$\log_2(B) + \lambda \approx 128$ bits thanks to our mod M procedure

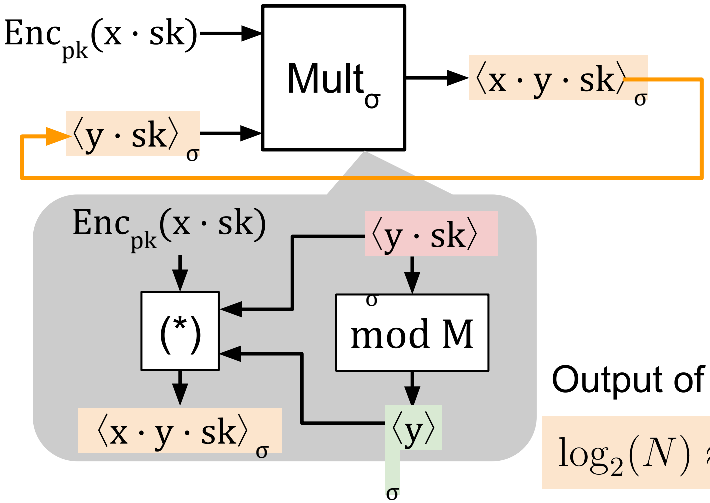
Output of (*) is:

$\log_2(N) \approx 3072$ bits

Time to compute modular exponentiation scales ~ **linearly** with the **length of the exponent**

Our Optimization: Speed Up (*) By Shortening Exponent

Recall our scheme



Running time of (*) ≈ two modular exponentiations
More precisely, this is the bottleneck:

$$ct_0^{\langle y \cdot sk \rangle_\sigma} \cdot ct_1^{\langle y \rangle_\sigma} \pmod{N^2}$$

where we unpack $\mathrm{Enc}_{pk}(x \cdot sk) = (ct_0, ct_1)$

3072 bits

$\log_2(B) + \lambda \approx 128$ bits thanks to our mod M procedure

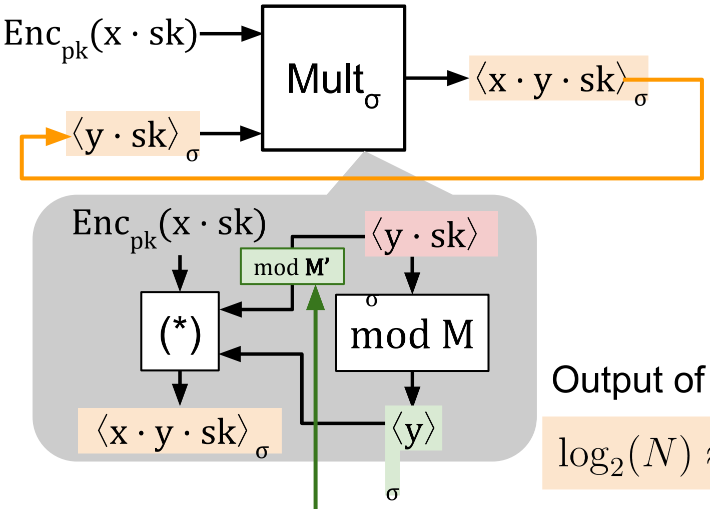
Output of (*) is:

$\log_2(N) \approx 3072$ bits

Time to compute modular exponentiation scales ~ **linearly** with the **length of the exponent**

Our Optimization: Speed Up (*) By Shortening Exponent

Recall our scheme



Running time of (*) ≈ two modular exponentiations
More precisely, this is the bottleneck:

$$ct_0^{\langle y \cdot sk\rangle_\sigma} \cdot ct_1^{\langle y\rangle_\sigma} \pmod{N^2}$$

where we unpack $\mathrm{Enc}_{pk}(x \cdot sk) = (ct_0, ct_1)$

3072 bits

$\log_2(B) + \lambda \approx 128$ bits thanks to our mod M procedure

Output of (*) is:

$\log_2(N) \approx 3072$ bits

Time to compute modular exponentiation scales ~ **linearly** with the **length of the exponent**

Idea: Use the same "mod M" trick, but use a larger modulus **M'**

Recall: Simplifying memory shares **(this work)**

Notation

$x = \langle x \rangle_A - \langle x \rangle_B$

Goal: derive $\langle y \rangle_\sigma$ locally from $\langle y \cdot sk \rangle_\sigma$

Pick $M \leq N \cdot 2^{-\lambda}$

$r \leftarrow\$ \{0, \dots, N-1\}$

$$\langle y \cdot sk \rangle_A = r + y \cdot sk \xrightarrow{\bmod M} = r' + y \cdot \cancel{sk} \pmod M$$

$$\boxed{sk \equiv 1 \pmod M}$$

$$\langle y \cdot sk \rangle_B = r \xrightarrow{\bmod M} = r'$$

$r' \approx_s \{0, \dots, M-1\}$

$r' := r \bmod M$

Shortening memory shares **(this work)**

Notation

$x = \langle x \rangle_A - \langle x \rangle_B$

Goal: derive a shorter share $\langle y \cdot sk \rangle_{\sigma'}$, of length $\log_2(\mathbf{M'})$ bits.

Pick $\mathbf{M'} \leq N \cdot 2^{-\lambda}$

$r \leftarrow\$ \{0, \ldots, N-1\}$

$\langle y \cdot sk \rangle_A$

$= r + y \cdot sk$

$\big\downarrow \bmod \mathbf{M'}$

$= r' + y \cdot sk \pmod{\mathbf{M'}}$

$\langle y \cdot sk \rangle_B$

$= r$

$\big\downarrow \bmod \mathbf{M'}$

$= r'$

$\boxed{sk \not\equiv 1 \pmod{\mathbf{M'}}}$

$\boxed{sk \equiv 1 \pmod{M}}$

$r' \approx_s \{0, \ldots, \mathbf{M'} - 1\}$

$r' := r \bmod \mathbf{M'}$

Shortening memory shares **(this work)**

Notation

$x = \langle x \rangle_A - \langle x \rangle_B$

Goal: derive a shorter share $\langle y \cdot sk \rangle_{\sigma'}$, of length $\log_2(\mathbf{M'})$ bits.

Pick $\mathbf{M'} = \max(y) \cdot \max(sk) \cdot 2^\lambda$

Need $y \cdot sk \cdot 2^\lambda \leq \mathbf{M'} \leq N \cdot 2^{-\lambda}$

$r \leftarrow\$ \{0, \ldots, N-1\}$

$\langle y \cdot sk \rangle_A$

$= r + y \cdot sk$

$\big\downarrow \bmod \mathbf{M'}$

$= r' + y \cdot sk \pmod{\mathbf{M'}}$

$= r' + y \cdot sk$ (over $\mathbb{Z}$) $= \langle y \cdot sk \rangle_A$

$\langle y \cdot sk \rangle_B$

$= r$

$\big\downarrow \bmod \mathbf{M'}$

$= r' = \langle y \cdot sk \rangle_B$

holds if $r' < \mathbf{M'} - y \cdot sk$, which is true with probability $\geq 1 - 2^{-\lambda}$

$r' \approx_s \{0, \ldots, \mathbf{M'} - 1\}$

$r' := r \bmod \mathbf{M'}$

Shortening memory shares **(this work)**

Notation
$x = \langle x \rangle_A - \langle x \rangle_B$

Goal: derive a shorter share $\langle y \cdot sk \rangle_\sigma$, of length $\log_2(\mathbf{M'})$ bits.

Pick $\mathbf{M'} = \max(y) \cdot \max(sk) \cdot 2^\lambda$

$|sk| \leq B^2 \cdot 2^{6\lambda}$, $\quad B := \max(y)$
Can set $\mathbf{M' = B^3 \cdot 2^{7\lambda}}$
$B = 1, \lambda = 128 \Rightarrow M'$ is 896 bits
(cf. N is 3072 bits)

$\langle y \cdot sk \rangle_A$

$= r + y \cdot sk$

$\downarrow \bmod \mathbf{M'}$

$= r' + y \cdot sk \pmod{\mathbf{M'}}$

$= r' + y \cdot sk$ (over $\mathbb{Z}$) $= \langle y \cdot sk \rangle_A$

holds if $r' < \mathbf{M'} - y \cdot sk$, which is true with probability $\geq 1 - 2^{-\lambda}$

$r' \approx_s \{0, \dots, \mathbf{M'} - 1\}$

$r' := r \bmod \mathbf{M'}$

$\langle y \cdot sk \rangle_B$

$= r$

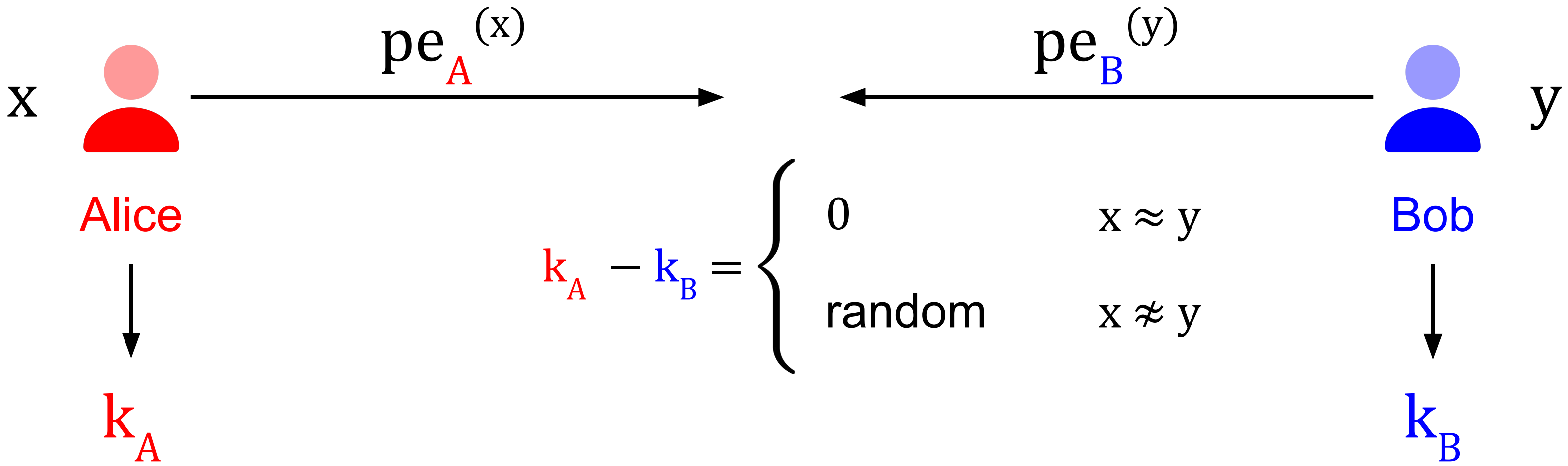
$\downarrow \bmod \mathbf{M'}$

$= r' = \langle y \cdot sk \rangle_B$

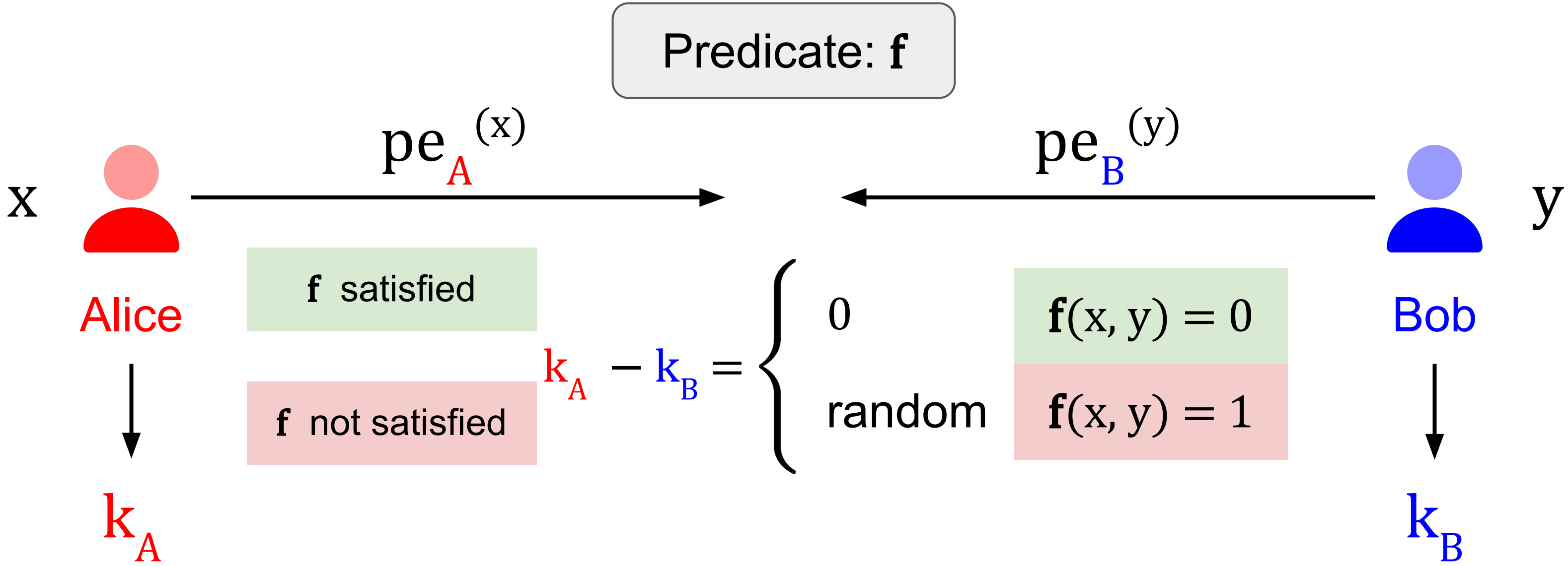
Roadmap

1. Overview of our work
2. MKHSS optimizations
3. Non-interactive conditional key exchange optimizations
4. Useful instantiations of key exchange
   a. Fuzzy PAKE
   b. Geolocation-based key exchange
5. Performance evaluation
6. Future works and conclusion

Non-Interactive Conditional Key Exchange [CDHJS'25]



Formalizing Non-Interactive Conditional Key Exchange [CDHJS'25]



Naïve Attempt 1

$\Delta \xleftarrow{\$} \{1, \ldots, 2^{\lambda} - 1\}$

Predicate: f

x, Δ Alice $\xrightarrow{pe_A^{(x,\Delta)}}$ $\xleftarrow{pe_B^{(y)}}$ Bob y

$$k_A - k_B = \begin{cases} 0 & f(x,y) = 0 \\ \Delta & f(x,y) = 1 \end{cases}$$

$k_A = \langle z \rangle_A$

$k_B = \langle z \rangle_B$

HSS Output

$z = f(x,y) \cdot \Delta$

Naïve Attempt 1

$\Delta \leftarrow\$ \{1, \ldots, 2^{\lambda} - 1\}$

Predicate: f

x, Δ — $pe_A^{(x, \Delta)}$ →

← $pe_B^{(y)}$ — y

Alice

Bob

$$k_A - k_B = \begin{cases} 0 & f(x, y) = 0 \\ \Delta & f(x, y) = 1 \end{cases}$$

$k_A = \langle z \rangle_A$

$k_B = \langle z \rangle_B$

Attack

Even if predicate **is not satisfied** $(f(x,y) = 1)$, Alice can still compute:

$k_B = k_A - \Delta$

HSS Output

$z = f(x,y) \cdot \Delta$

Naïve Attempt 1

$\Delta \leftarrow_{\$} \{1, \dots, 2^{\lambda} - 1\}$

Predicate: f

x, Δ

Alice

$pe_A^{(x, \Delta)}$

$pe_B^{(y)}$

Bob

y

$$k_A - k_B = \begin{cases} 0 & f(x, y) = 0 \\ \Delta & f(x, y) = 1 \end{cases}$$

$k_A = \langle z \rangle_A$

Does not look random to Alice, who could be the adversary!

$k_B = \langle z \rangle_B$

HSS Output

$z = f(x,y) \cdot \Delta$

Naïve Attempt 2

Predicate: f

$\Delta_A \leftarrow \$ \{1, \ldots, 2^\lambda - 1\}$

$\Delta_B \leftarrow \$ \{1, \ldots, 2^\lambda - 1\}$

x, Δ_A Alice $\xrightarrow{pe_A^{(x, \Delta_A)}}$

$\xleftarrow{pe_B^{(y, \Delta_B)}}$ Bob y, Δ_B

$$k_A - k_B = \begin{cases} 0 & f(x, y) = 0 \\ \Delta_A \cdot \Delta_B & f(x, y) = 1 \end{cases}$$

Alice: $k_A = \langle z \rangle_A$

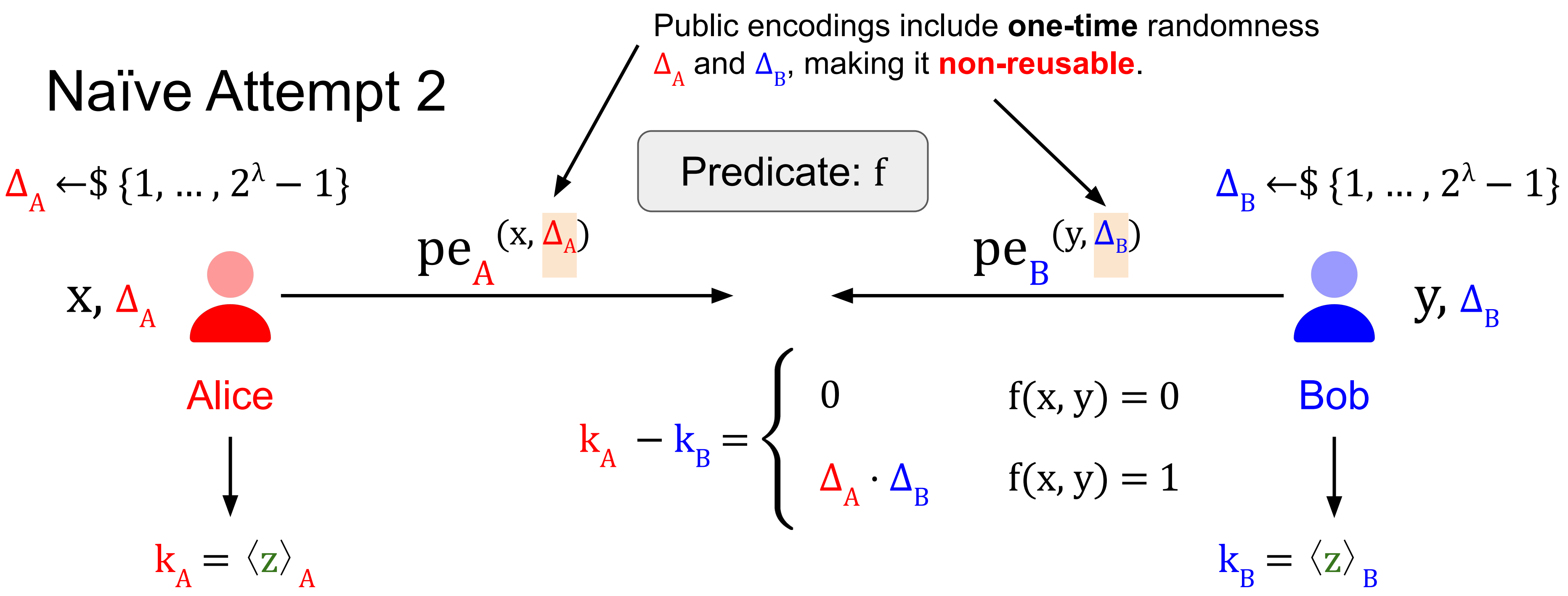
Bob: $k_B = \langle z \rangle_B$

✅ Looks random to both parties.

HSS Output

$$z = f(x,y) \cdot \Delta_A \cdot \Delta_B$$

Naïve Attempt 2



HSS Output

$$z = f(x,y) \cdot \Delta_A \cdot \Delta_B$$

Naïve Attempt 2

Public encodings include **one-time** randomness Δ_A and Δ_B, making it **non-reusable**.

Predicate: f

$\Delta_A \leftarrow\$ \{1, \ldots, 2^{\lambda} - 1\}$

$\Delta_B \leftarrow\$ \{1, \ldots, 2^{\lambda} - 1\}$

$\mathrm{pe}_A^{(x, \Delta_A)}$

$\mathrm{pe}_B^{(y, \Delta_B)}$

x, Δ_A

Alice

y, Δ_B

Bob

$$k_A - k_B = \begin{cases} 0 & f(x, y) = 0 \\ \Delta_A \cdot \Delta_B & f(x, y) = 1 \end{cases}$$

$k_A = \langle z \rangle_A$

$k_B = \langle z \rangle_B$

A truly **non-interactive** conditional key exchange need reusability of pe_A, pe_B

HSS Output

$$z = f(x,y) \cdot \Delta_A \cdot \Delta_B$$

Solution by Prior Work: Use a PRF [CDHJS'25]

Predicate: f

$K_A \leftarrow \mathrm{PRF.Gen}(1^\lambda)$

$K_B \leftarrow \mathrm{PRF.Gen}(1^\lambda)$

x, K_A

Alice

$\mathrm{pe}_A^{(x, K_A)}$

$\mathrm{pe}_B^{(y, K_B)}$

y, K_B

Bob

$$k_A - k_B = \begin{cases} 0 & f(x, y) = 0 \\ \Delta_A \cdot \Delta_B & f(x, y) = 1 \end{cases}$$

$k_A = \langle z \rangle_A$

$k_B = \langle z \rangle_B$

HSS Program

$$\Delta_A = \mathrm{PRF.Eval}(K_A, id_A \,\|\, id_B \,\|\, f)$$

$$\Delta_B = \mathrm{PRF.Eval}(K_B, id_A \,\|\, id_B \,\|\, f)$$

$$z = f(x,y) \cdot \Delta_A \cdot \Delta_B$$

Solution by Prior Work: Use a PRF [CDHJS'25]

Predicate: f

$K_A \leftarrow \mathrm{PRF.Gen}(1^\lambda)$ $\quad$ $K_B \leftarrow \mathrm{PRF.Gen}(1^\lambda)$

x, K_A — Alice $\xrightarrow{\mathrm{pe}_A^{(x, K_A)}}$ $\quad$ $\xleftarrow{\mathrm{pe}_B^{(y, K_B)}}$ Bob — y, K_B

$$k_A - k_B = \begin{cases} 0 & f(x,y) = 0 \\ \Delta_A \cdot \Delta_B & f(x,y) = 1 \end{cases}$$

Alice $\downarrow$ $k_A = \langle z \rangle_A$

Bob $\downarrow$ $k_B = \langle z \rangle_B$

HSS Program

$$\Delta_A = \mathrm{PRF.Eval}(K_A, \mathrm{id}_A \,\|\, \mathrm{id}_B \,\|\, f)$$

$$\Delta_B = \mathrm{PRF.Eval}(K_B, \mathrm{id}_A \,\|\, \mathrm{id}_B \,\|\, f)$$

$$z = f(x,y) \cdot \Delta_A \cdot \Delta_B$$

Evaluating an NC^1 PRF inside HSS is **concretely expensive**!

Our Optimization #1: Use a Hash Outside of HSS Instead

Predicate: f

$\Delta_A \leftarrow\$ \{1, \ldots, 2^\lambda - 1\}$

$\Delta_B \leftarrow\$ \{1, \ldots, 2^\lambda - 1\}$

x, Δ_A Alice $\xrightarrow{\mathrm{pe}_A^{(x)}}$

$\xleftarrow{\mathrm{pe}_B^{(y)}}$ Bob y, Δ_B

$k_A = k_B$ $\quad f(x, y) = 0$

k_A and k_B are independent $\quad f(x, y) = 1$

$k_A = \mathbf{H}(\langle z \rangle_A \parallel \mathrm{id}_A \parallel \mathrm{id}_B \parallel f)$

$k_B = \mathbf{H}(\langle z \rangle_B \parallel \mathrm{id}_A \parallel \mathrm{id}_B \parallel f)$

HSS Output

$z = f(x,y) \cdot \Delta_A \cdot \Delta_B$

Our Optimization #1: Use a Hash Outside of HSS Instead

Predicate: f

$\Delta_A \leftarrow\$ \{1, \dots, 2^{\lambda} - 1\}$

$\Delta_B \leftarrow\$ \{1, \dots, 2^{\lambda} - 1\}$

x, Δ_A Alice $\xrightarrow{pe_A^{(x)}}$

$\xleftarrow{pe_B^{(y)}}$ Bob y, Δ_B

$$\begin{cases} k_A = k_B & f(x, y) = 0 \\ k_A \text{ and } k_B \text{ are independent} & f(x, y) = 1 \end{cases}$$

$k_A = \mathbf{H}(\langle z \rangle_A \,\|\, aux)$

$k_B = \mathbf{H}(\langle z \rangle_B \,\|\, aux)$

HSS Output

$z = f(x,y) \cdot \Delta_A \cdot \Delta_B$

$aux := id_A \,\|\, id_B \,\|\, f$

Inefficiency: Need HSS to Support Large Integer Values

$\Delta_A \leftarrow \$ \{1, \ldots, 2^{\lambda} - 1\}$

Predicate: f

$\Delta_B \leftarrow \$ \{1, \ldots, 2^{\lambda} - 1\}$

x, Δ_A Alice $\xrightarrow{\mathrm{pe}_A^{(x)}}$ $\xleftarrow{\mathrm{pe}_B^{(y)}}$ Bob y, Δ_B

$$\begin{cases} k_A = k_B & f(x, y) = 0 \\ k_A \text{ and } k_B \text{ are independent} & f(x, y) = 1 \end{cases}$$

$k_A = \mathbf{H}(\langle z \rangle_A \,\|\, \mathrm{aux})$

$k_B = \mathbf{H}(\langle z \rangle_B \,\|\, \mathrm{aux})$

HSS Output

$z = f(x,y) \cdot \Delta_A \cdot \Delta_B \in \mathbf{\{0, \ldots, 2^{2\lambda} - 1\}}$

$\mathrm{aux} := \mathrm{id}_A \,\|\, \mathrm{id}_B \,\|\, f$

Inefficiency: Need HSS to Support Large Integer Values

Predicate: f

$\Delta_A \leftarrow\$ \{1, \ldots, 2^{\lambda} - 1\}$

$\Delta_B \leftarrow\$ \{1, \ldots, 2^{\lambda} - 1\}$

x, Δ_A — Alice $\xrightarrow{pe_A^{(x)}}$

$\xleftarrow{pe_B^{(y)}}$ Bob — y, Δ_B

$$\begin{cases} k_A = k_B & f(x, y) = 0 \\ k_A \text{ and } k_B \text{ are independent} & f(x, y) = 1 \end{cases}$$

$k_A = \mathbf{H}(\langle z \rangle_A \,\|\, \mathrm{aux})$

$k_B = \mathbf{H}(\langle z \rangle_B \,\|\, \mathrm{aux})$

Need to set $\mathbf{B = 2^{2\lambda}}$, reducing concrete efficiency

HSS Output

$\mathrm{aux} := \mathrm{id}_A \,\|\, \mathrm{id}_B \,\|\, f$

$z = f(x,y) \cdot \Delta_A \cdot \Delta_B \in \mathbf{\{0, \ldots, 2^{2\lambda} - 1\}}$

Our Optimization #2: Exploit Memory Share Structure

Predicate: f

$\Delta_A \leftarrow\$ \{1, \ldots, 2^{\lambda} - 1\}$

$\Delta_B \leftarrow\$ \{1, \ldots, 2^{\lambda} - 1\}$

x, Δ_A Alice $\xrightarrow{pe_A^{(x)}}$ $\xleftarrow{pe_B^{(y)}}$ Bob y, Δ_B

$$\begin{cases} k_A = k_B & f(x, y) = 0 \\ k_A \text{ and } k_B \text{ are independent} & f(x, y) = 1 \end{cases}$$

$k_A = \mathbf{H}(\langle z' \cdot \Delta_A \cdot \Delta_B \rangle_A \,\|\, aux)$

$k_B = \mathbf{H}(\langle z' \cdot \Delta_A \cdot \Delta_B \rangle_B \,\|\, aux)$

Function Output

$z' = f(x,y)$

$aux := id_A \,\|\, id_B \,\|\, f$

Our Optimization #2: Exploit Memory Share Structure

Predicate: f

x Alice $\xrightarrow{pe_A^{(x)}}$ $\xleftarrow{pe_B^{(y)}}$ Bob y

$$\begin{cases} k_A = k_B & f(x,y) = 0 \\ k_A \text{ and } k_B \text{ are independent} & f(x,y) = 1 \end{cases}$$

$k_A = \mathbf{H}(\langle z' \cdot sk_A \cdot sk_B \rangle_A \,\|\, aux)$

$k_B = \mathbf{H}(\langle z' \cdot sk_A \cdot sk_B \rangle_B \,\|\, aux)$

Use sk_A, sk_B in place of Δ_A, Δ_B

Function Output

$z' = f(x,y)$

$aux := id_A \,\|\, id_B \,\|\, f$

Our Optimization #2: Exploit Memory Share Structure

Predicate: f

x Alice $\xrightarrow{pe_A^{(x)}}$ $\xleftarrow{pe_B^{(y)}}$ Bob y

$$\begin{cases} k_A = k_B & f(x, y) = 0 \\ k_A \text{ and } k_B \text{ are independent} & f(x, y) = 1 \end{cases}$$

$k_A = \mathbf{H}(\langle z' \cdot sk \rangle_A \,||\, aux)$

$k_B = \mathbf{H}(\langle z' \cdot sk \rangle_B \,||\, aux)$

Joint secret key: $sk = sk_A \cdot sk_B$

Function Output

$z' = f(x,y)$

$aux := id_A \,||\, id_B \,||\, f$

Our Optimization #2: Exploit Memory Share Structure

Predicate: f

x Alice $\xrightarrow{pe_A^{(x)}}$ $\xleftarrow{pe_B^{(y)}}$ Bob y

$$\begin{cases} k_A = k_B & f(x,y) = 0 \\ k_A \text{ and } k_B \text{ are independent} & f(x,y) = 1 \end{cases}$$

$k_A = \mathbf{H}(\langle z' \cdot sk \rangle_A \,\|\, aux)$

$k_B = \mathbf{H}(\langle z' \cdot sk \rangle_B \,\|\, aux)$

Memory Share of z'

MKHSS Output

$z' = f(x,y)$

$aux := id_A \,\|\, id_B \,\|\, f$

Our Optimization #2: Exploit Memory Share Structure

Predicate: f

x Alice $\xrightarrow{pe_A^{(x)}}$ $\xleftarrow{pe_B^{(y)}}$ Bob y

$$\begin{cases} k_A = k_B & f(x, y) = 0 \\ k_A \text{ and } k_B \text{ are independent} & f(x, y) = 1 \end{cases}$$

$k_A = \mathbf{H}(\langle z' \cdot sk \rangle_A \,\|\, aux)$

$k_B = \mathbf{H}(\langle z' \cdot sk \rangle_B \,\|\, aux)$

Memory Share of z'

MKHSS Output

$z' = f(x,y) \in \{0,1\}$

✅ Output is a single bit: $\mathbf{B = 1}$ is possible

$aux := id_A \,\|\, id_B \,\|\, f$

Roadmap

1. Overview of our work
2. MKHSS optimizations
3. Non-interactive conditional key exchange optimizations
4. Useful instantiations of key exchange
    a. Fuzzy PAKE
    b. Geolocation-based key exchange
5. Performance evaluation
6. Future works and conclusion

Prior work requires 5+ rounds of interaction

Concrete Instantiation: fPAKE [DHPRY'18]

fPAKE: fuzzy password-authenticated key exchange

Predicate: $f(x,y) = 1 \Leftrightarrow \mathrm{EditDistance}(x, y) \leq T$

correct horse battery staple $= x$

$y =$ corrupt hose buttery stable

$pe_A^{(x)} \longrightarrow$

$\longleftarrow pe_B^{(y)}$

Alice

Bob

$\begin{cases} k_A = k_B & f(x, y) = 0 \\ \text{Independent } k_A \text{ and } k_B & f(x, y) = 1 \end{cases}$

k_A

k_B

Concrete Instantiation: fPAKE

fPAKE: fuzzy password-authenticated key exchange

Predicate: $\mathbf{f}(x,y) = 1 \Leftrightarrow \mathrm{EditDistance}(x, y) \leq T$

correct horse battery staple $= x$

$y =$ corrupt hose buttery stable

$pe_A^{(x)}$ →

← $pe_B^{(y)}$

Alice

Bob

$$\begin{cases} k_A = k_B & f(x, y) = 0 \\ \text{Independent } k_A \text{ and } k_B & f(x, y) = 1 \end{cases}$$

k_A

k_B

Technical challenge: Implement **f** as an **RMS** program: every multiplication must take an input as an operand

Concrete Instantiation: fPAKE

fPAKE: **fuzzy** password-authenticated key exchange

correct
horse
battery
staple $= x$

Predicate: $\mathbf{f}(x,y) = 1 \Leftrightarrow \text{EditDistance}(x, y) \leq T$

$y =$ corrupt
hose
buttery
stable

$pe_A^{(x)} \longrightarrow$

$\longleftarrow pe_B^{(y)}$

Alice

Bob

$$k_A \quad \quad k_B$$

$$\begin{cases} k_A = k_B & f(x, y) = 0 \\ \text{Independent } k_A \text{ and } k_B & f(x, y) = 1 \end{cases}$$

Technical challenge: Implement **f** as an **RMS** program: every multiplication must take an input as an operand

RMS $(B = 1)$ is sufficient to compute **branching programs**! [BGI'16]

Our contribution: compute useful fuzziness metrics

Useful fuzziness metric: Hamming distance

```
x   = correct horse battery staple
y   = corrupt house buttery stable
```

$$(\mathrm{HD}(x, y) = 4)$$

$$x \approx y \Leftrightarrow \mathrm{HD}(x, y) \leq T$$

Our contribution: compute useful fuzziness metrics

Useful fuzziness metric: Hamming distance

```
x   = correct horse battery staple
y   = corrupt house buttery stable
```

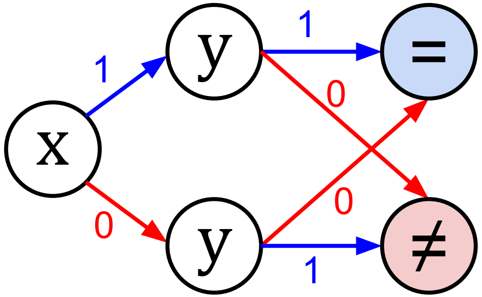
$$(\mathrm{HD}(x, y) = 4)$$

$$x \approx y \Leftrightarrow \mathrm{HD}(x, y) \leq T$$

As a warmup, we show how to compute this for **binary** strings.

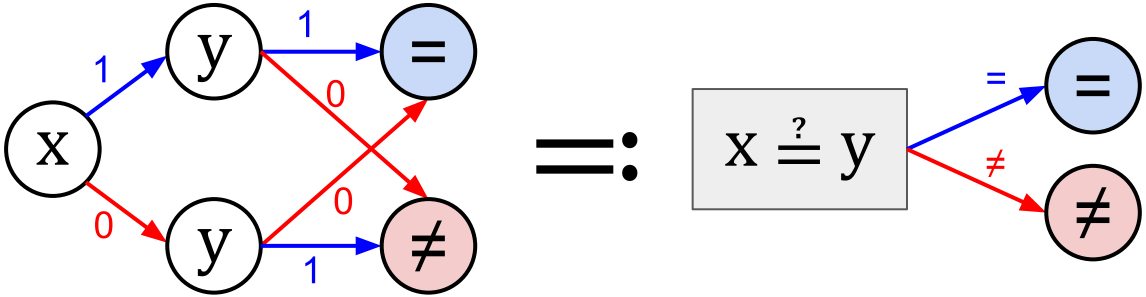
Warmup: branching program for binary Hamming distance

Bit equality: $x \stackrel{?}{=} y$



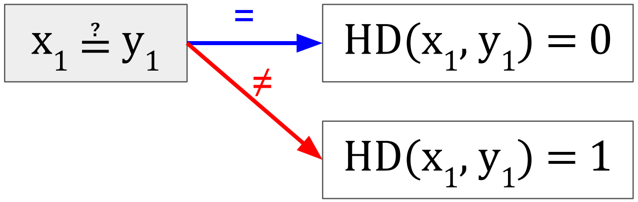
Warmup: branching program for binary Hamming distance

Bit equality: $x \stackrel{?}{=} y$

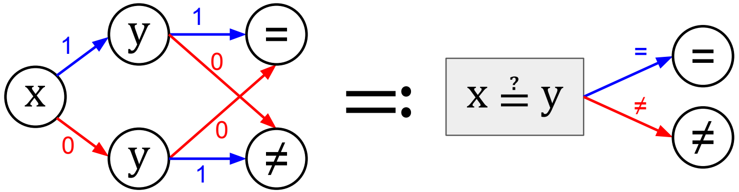


Warmup: branching program for binary Hamming distance

Threshold Hamming distance: $\mathrm{HD}(x, y) \overset{?}{\leq} T$

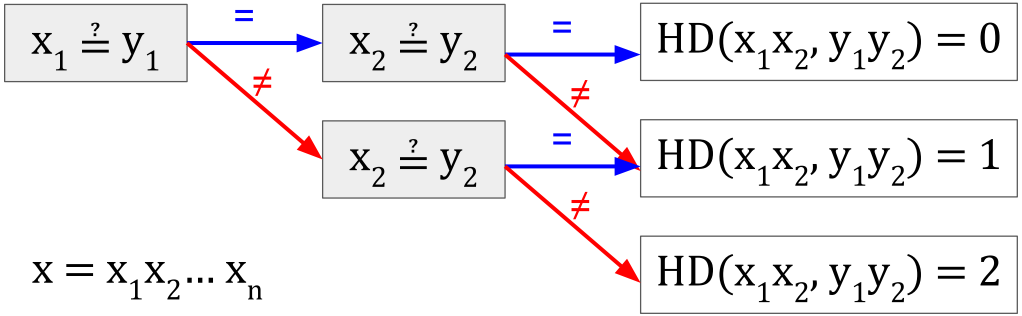


$$x = x_1 x_2 \ldots x_n$$
$$y = y_1 y_2 \ldots y_n$$



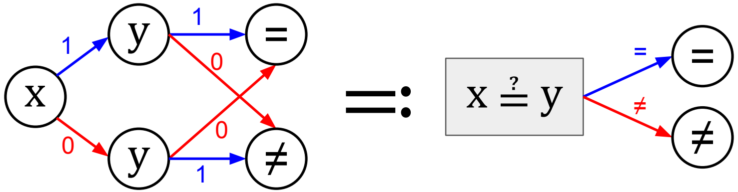
Warmup: branching program for binary Hamming distance

Threshold Hamming distance: $\mathrm{HD}(x, y) \overset{?}{\leq} T$



$x = x_1x_2 \ldots x_n$

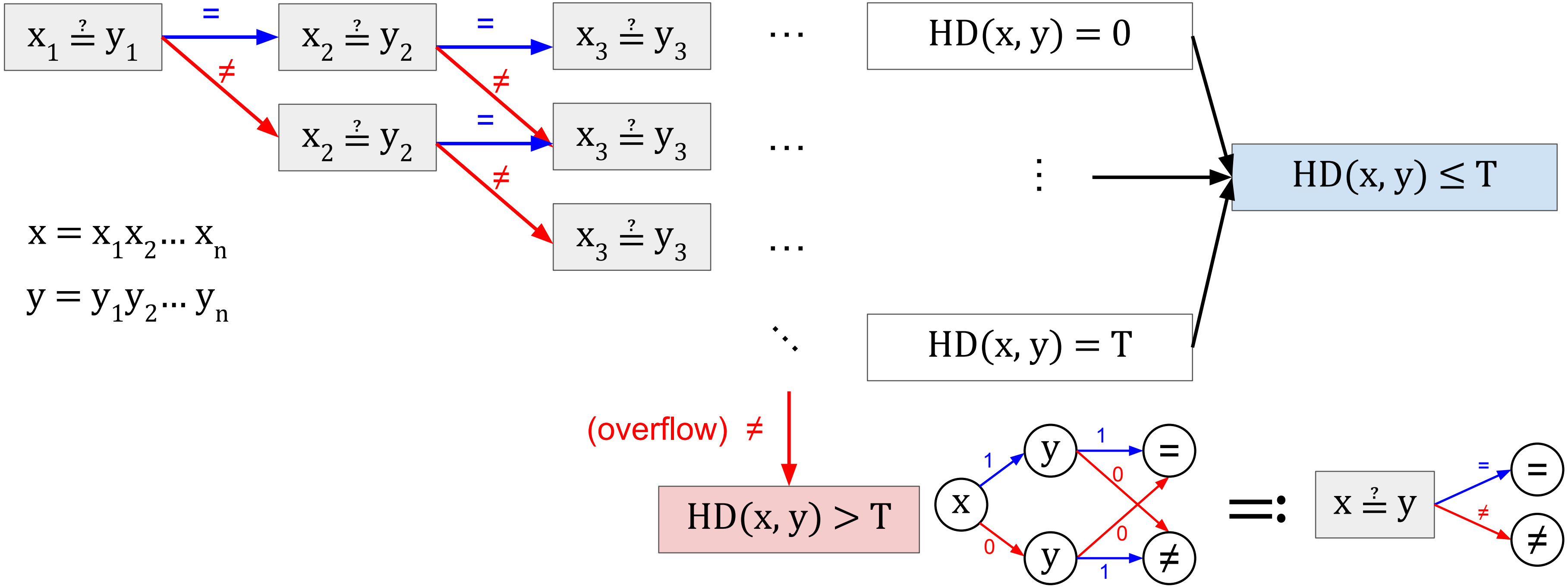
$y = y_1y_2 \ldots y_n$



Warmup: branching program for binary Hamming distance

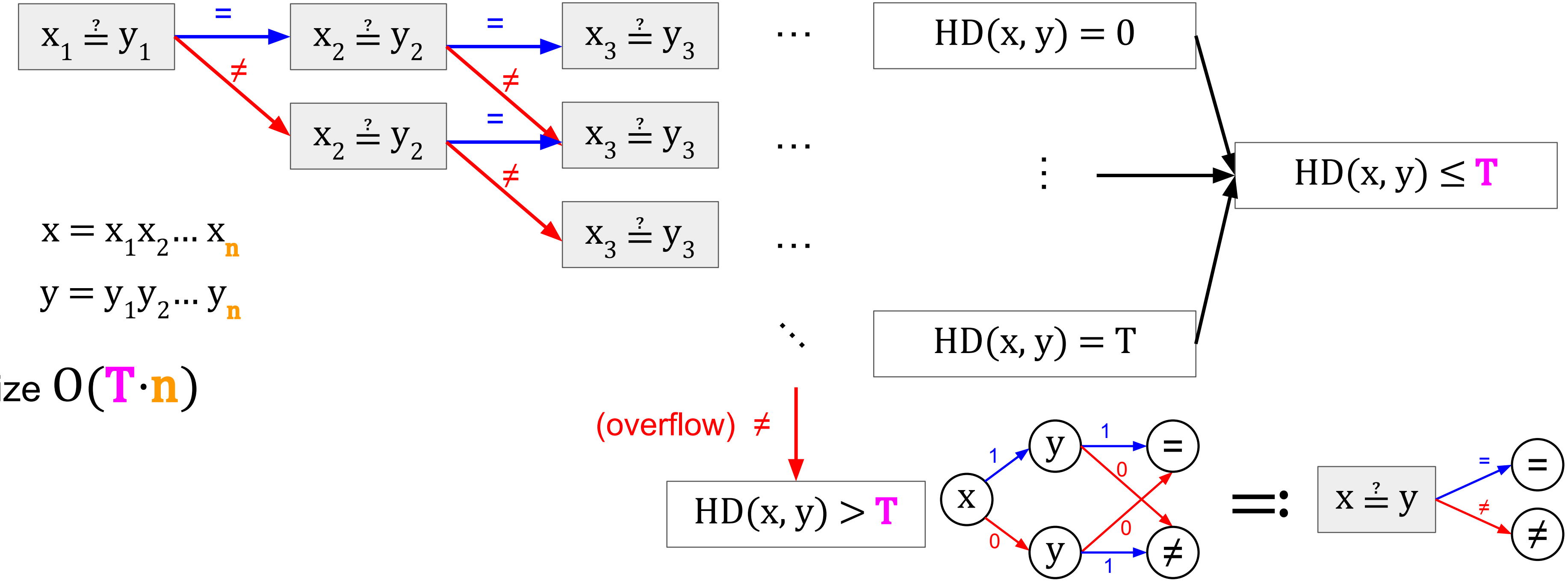
Threshold Hamming distance: $\mathrm{HD}(x, y) \overset{?}{\leq} T$

$$x = x_1 x_2 \dots x_n$$
$$y = y_1 y_2 \dots y_n$$



Warmup: branching program for binary Hamming distance

Threshold Hamming distance: $\mathrm{HD}(x, y) \overset{?}{\leq} T$



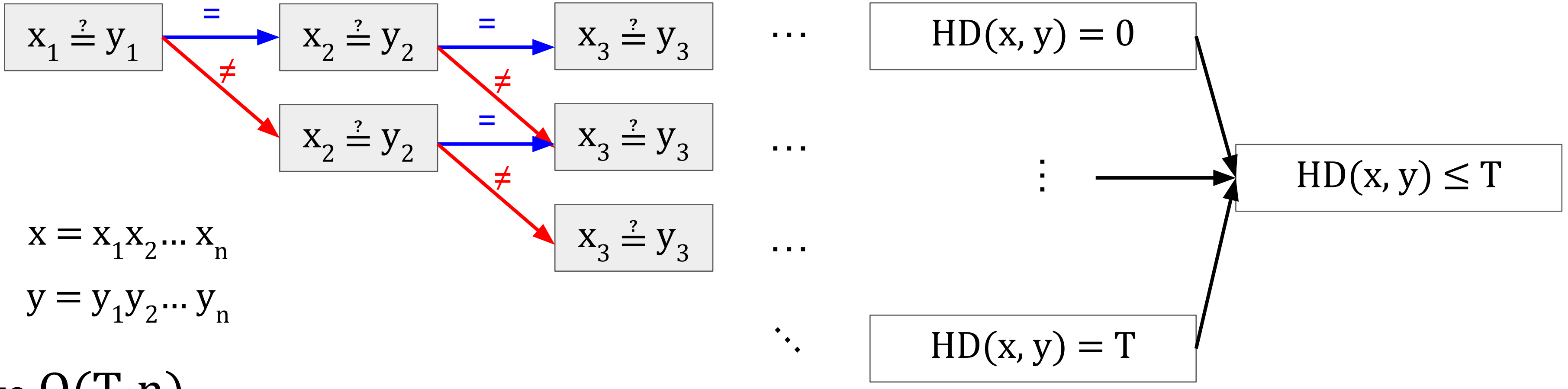
$x = x_1 x_2 \ldots x_n$

$y = y_1 y_2 \ldots y_n$

Size $O(T \cdot n)$

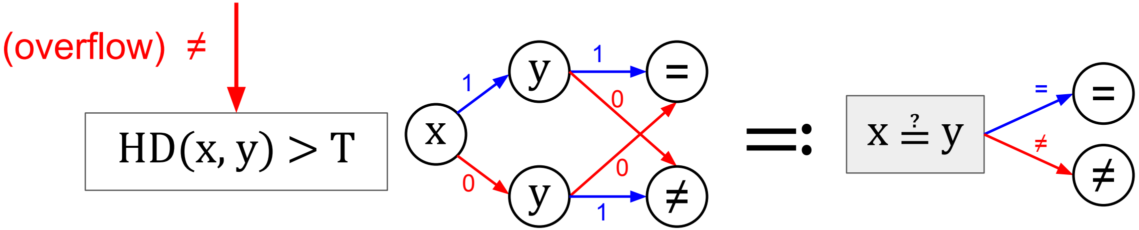
Warmup: branching program for binary Hamming distance

Threshold Hamming distance: $\mathrm{HD}(x, y) \overset{?}{\leq} T$



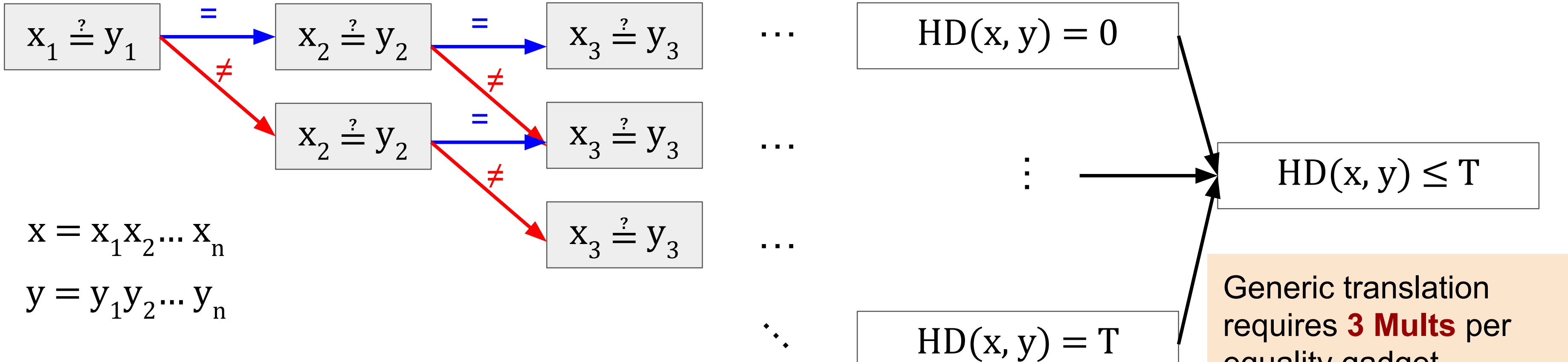
Size $O(T \cdot n)$

Evaluation cost is one **Mult (5 ms)** per **node**



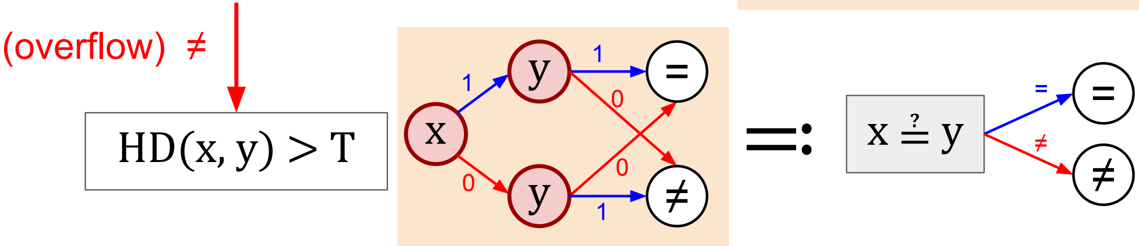
Warmup: branching program for binary Hamming distance

Threshold Hamming distance: $\mathrm{HD}(x, y) \overset{?}{\leq} T$



Size $O(T \cdot n)$

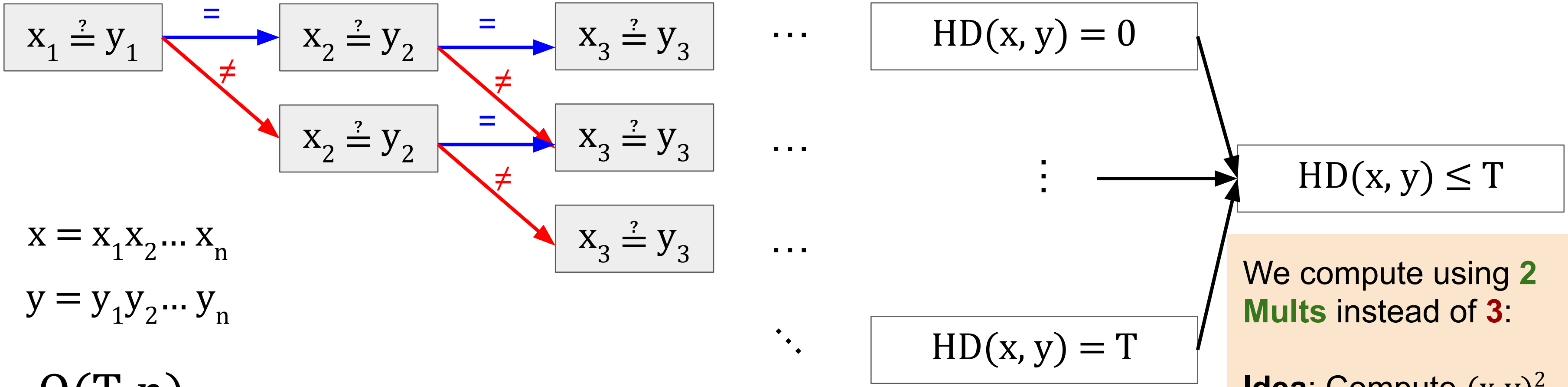
Evaluation cost is one **Mult (5 ms)** per **node** via generic translation from branching programs [BCGIO'17]

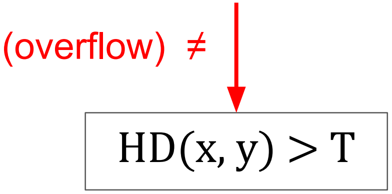


Warmup: branching program for binary Hamming distance

Threshold Hamming distance: $\mathrm{HD}(x, y) \overset{?}{\leq} T$

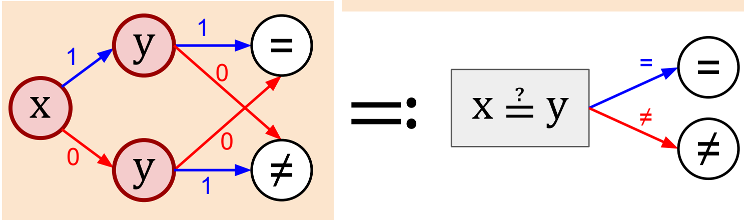
$$(x-y)^2 = \begin{cases} 0 & \text{if } x = y \\ 1 & \text{if } x \neq y \end{cases}$$





We compute using **2 Mults** instead of **3**:

Idea: Compute $(x-y)^2$

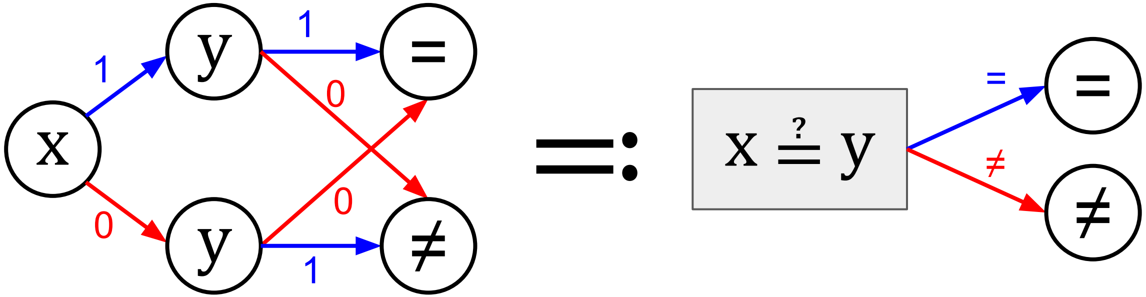


Size $O(T \cdot n)$

Evaluation cost is one **Mult (5 ms)** per **node** via generic translation from branching programs [BCGIO'17]

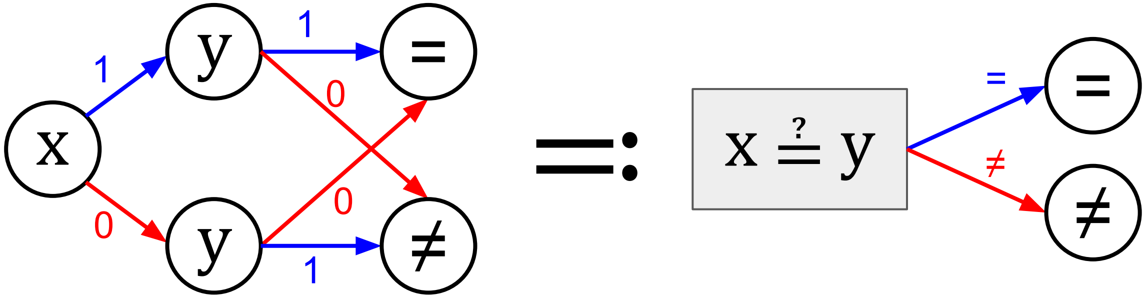
Generalization: Hamming distance over any alphabet

Recall bit equality: $x \overset{?}{=} y$

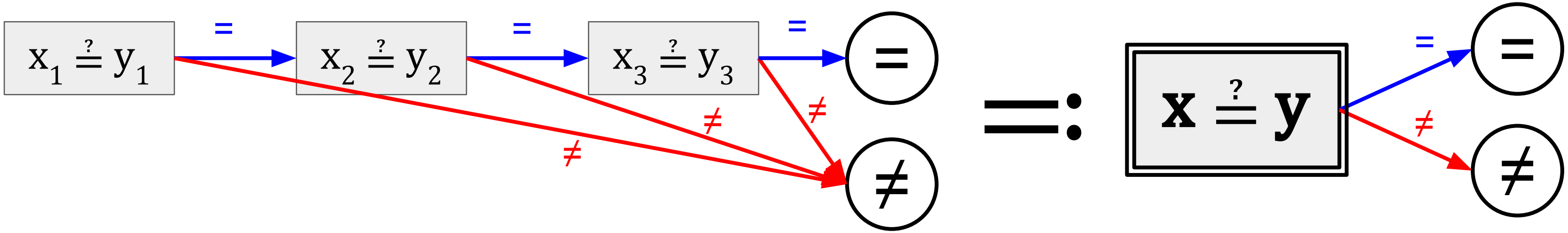


Generalization: Hamming distance over any alphabet

Recall bit equality: $x \overset{?}{=} y$



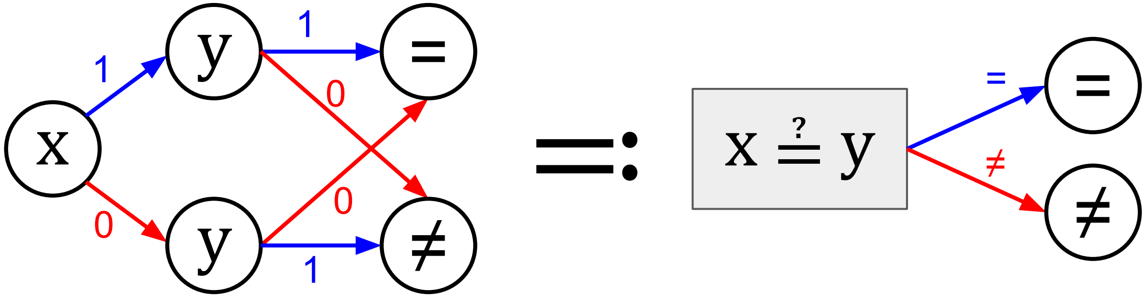
Character equality: $\mathbf{x} \overset{?}{=} \mathbf{y}$ (Encode each character in binary: $\mathbf{x} = x_1x_2x_3$)



Generalization: Hamming distance over any alphabet

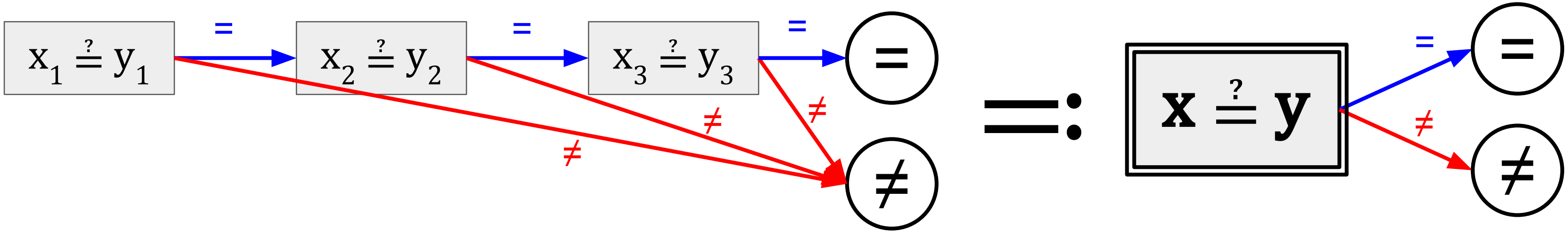
Recall bit equality: $x \stackrel{?}{=} y$

Can also compute a recursive notion of Hamming distance, which tolerates insertions and deletions better. (See Paper)



Character equality: $\mathbf{x} \stackrel{?}{=} \mathbf{y}$

(Encode each character in binary: $\mathbf{x} = x_1 x_2 x_3$)

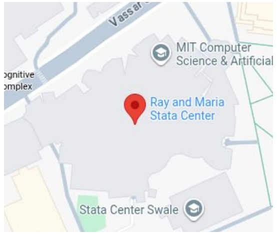


Roadmap

1. Overview of our work
2. MKHSS optimizations
3. Non-interactive conditional key exchange optimizations
4. Useful instantiations of key exchange
    a. Fuzzy PAKE
    b. Geolocation-based key exchange
5. Performance evaluation
6. Future works and conclusion

Implementing geolocation-based key exchange

$x =$

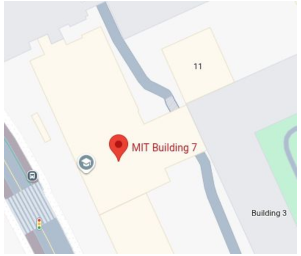


$\xrightarrow{\mathsf{pe}_A}$

$\xleftarrow{\mathsf{pe}_B}$

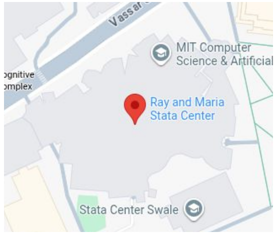
Distance Policy: P

$y =$



Implementing geolocation-based key exchange

$\vec{x} = (x_1, x_2)$

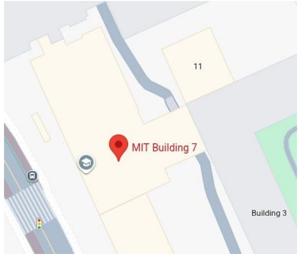


pe_A $\longrightarrow$

$\longleftarrow$ pe_B

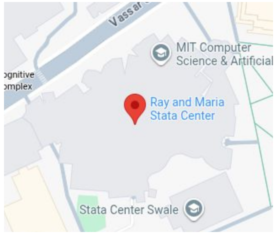
Distance Policy: P

$\vec{y} = (y_1, y_2)$



Implementing geolocation-based key exchange

$\vec{x} = (x_1, x_2)$

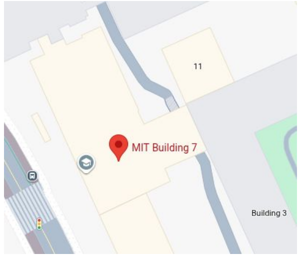


$\xrightarrow{\mathsf{pe}_A}$ $\xleftarrow{\mathsf{pe}_B}$

Distance Policy: P

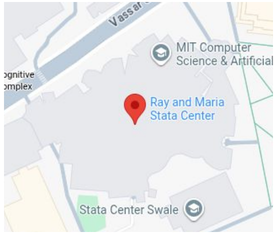
$$P(\vec{x}, \vec{y}) = 0 \iff \|\vec{x} - \vec{y}\|_\infty \leq d$$

$\vec{y} = (y_1, y_2)$



Implementing geolocation-based key exchange

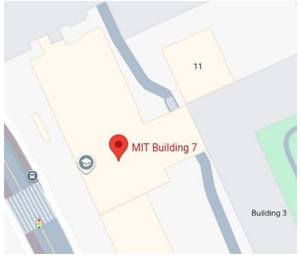
$\vec{x} = (x_1, x_2)$



pe_A

pe_B

$\vec{y} = (y_1, y_2)$



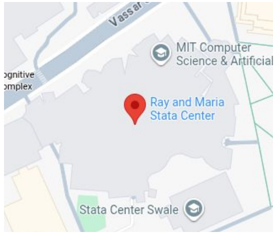
Distance Policy: P

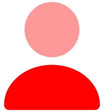
$$P(\vec{x}, \vec{y}) = 0 \iff \|\vec{x} - \vec{y}\|_\infty \leq d$$

$$P(\vec{x}, \vec{y}) = 0 \iff \max(|x_1 - y_1|, |x_2 - y_2|) \leq d$$

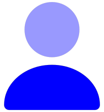
Implementing geolocation-based key exchange

$\vec{x} = (x_1, x_2)$

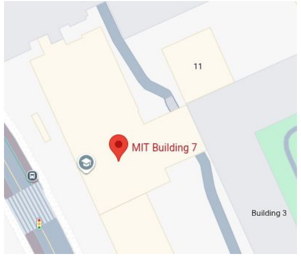




$\xrightarrow{\mathsf{pe}_A}$ $\xleftarrow{\mathsf{pe}_B}$



$\vec{y} = (y_1, y_2)$



Distance Policy: P

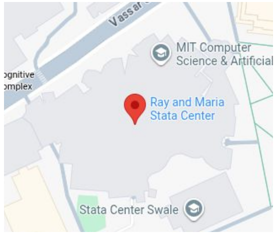
$$P(\vec{x}, \vec{y}) = 0 \iff \|\vec{x} - \vec{y}\|_\infty \leq d$$

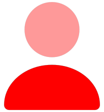
$$P(\vec{x}, \vec{y}) = 0 \iff \max(|x_1 - y_1|, |x_2 - y_2|) \leq d$$

$$\iff \begin{cases} |x_1 - y_1| \leq d \\ |x_2 - y_2| \leq d \end{cases}$$

Implementing geolocation-based key exchange

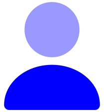
$\vec{x} = (x_1, x_2)$



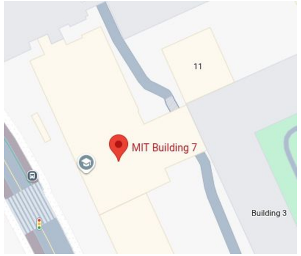


pe_A $\longrightarrow$

$\longleftarrow$ pe_B



$\vec{y} = (y_1, y_2)$



Distance Policy: P

$$P(\vec{x}, \vec{y}) = 0 \iff \|\vec{x} - \vec{y}\|_\infty \leq d$$

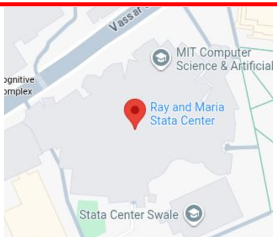
$$P(\vec{x}, \vec{y}) = 0 \iff \max(|x_1 - y_1|, |x_2 - y_2|) \leq d$$

$$\iff \begin{cases} |x_1 - y_1| \leq d \\ |x_2 - y_2| \leq d \end{cases}$$

$$\iff \begin{cases} y_1 - d \leq x_1 \leq y_1 + d \\ y_2 - d \leq x_2 \leq y_2 + d \end{cases}$$

Implementing geolocation-based key exchange

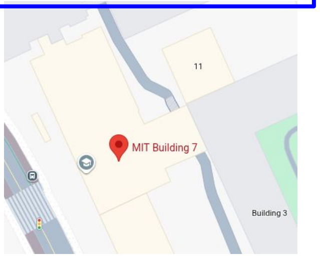
$\vec{x} = (x_1, x_2)$



pe_A →

← pe_B

$\vec{y} = (y_1, y_2)$



Distance Policy: P

$$P(\vec{x}, \vec{y}) = 0 \iff \|\vec{x} - \vec{y}\|_\infty \leq d$$

$$P(\vec{x}, \vec{y}) = 0 \iff \max(|x_1 - y_1|, |x_2 - y_2|) \leq d$$

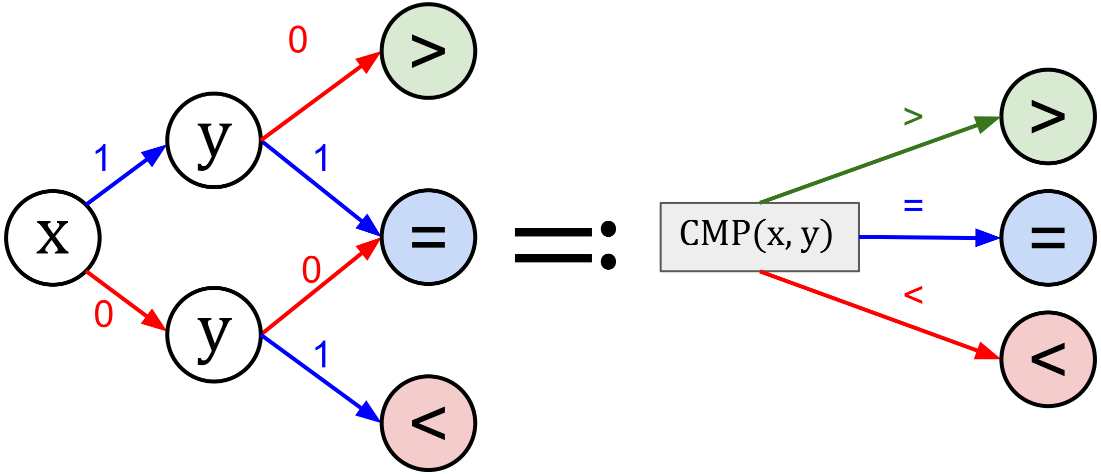
$$\iff \begin{cases} |x_1 - y_1| \leq d \\ |x_2 - y_2| \leq d \end{cases}$$

$$\iff \begin{cases} \boxed{y_1 - d} \leq \boxed{x_1} \leq \boxed{y_1 + d} \\ \boxed{y_2 - d} \leq \boxed{x_2} \leq \boxed{y_2 + d} \end{cases}$$

Reduces the problem to
four integer comparisons
on
inputs known to some party

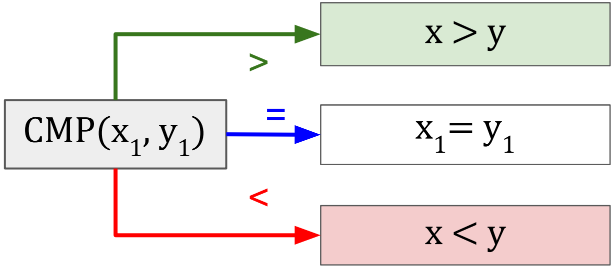
Branching Program for Integer Comparison

Bit comparison: $\mathrm{CMP}(x, y) \in \{<, =, >\}$



Branching Program for Integer Comparison

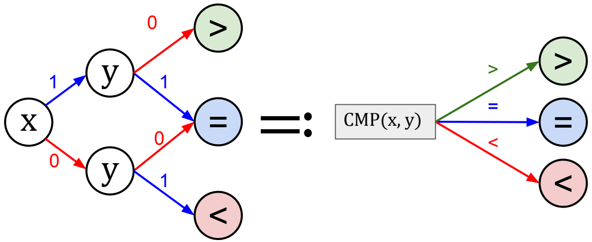
Comparing x and y.



Binary decompositions

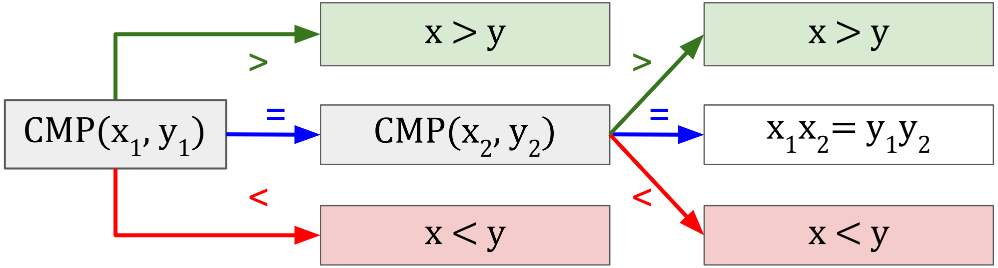
$\text{bit\_decomp}(x) = x_1 x_2 \ldots x_n$

$\text{bit\_decomp}(y) = y_1 y_2 \ldots y_n$



Branching Program for Integer Comparison

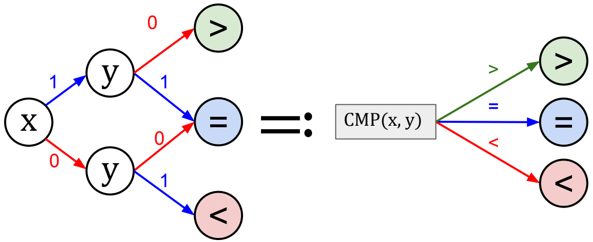
Comparing x and y.



Binary decompositions

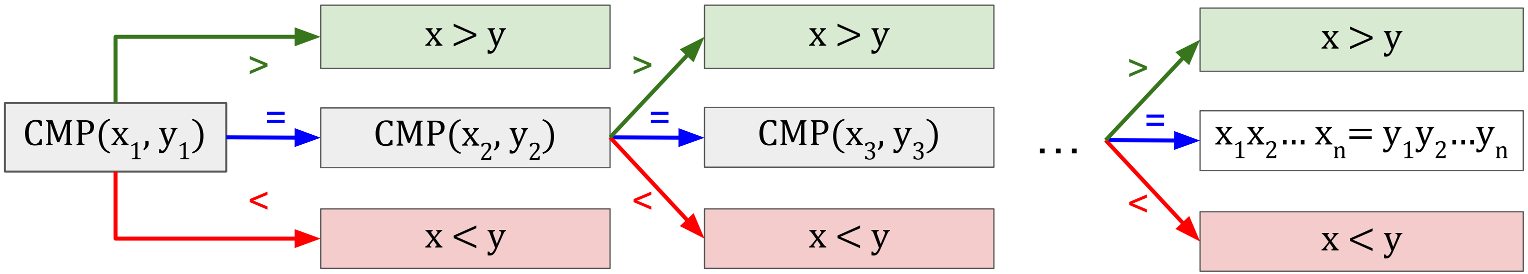
$\mathrm{bit\_decomp}(x) = x_1x_2\ldots x_n$

$\mathrm{bit\_decomp}(y) = y_1y_2\ldots y_n$



Branching Program for Integer Comparison

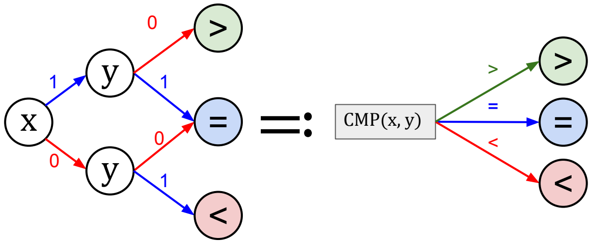
Comparing x and y.



Binary decompositions

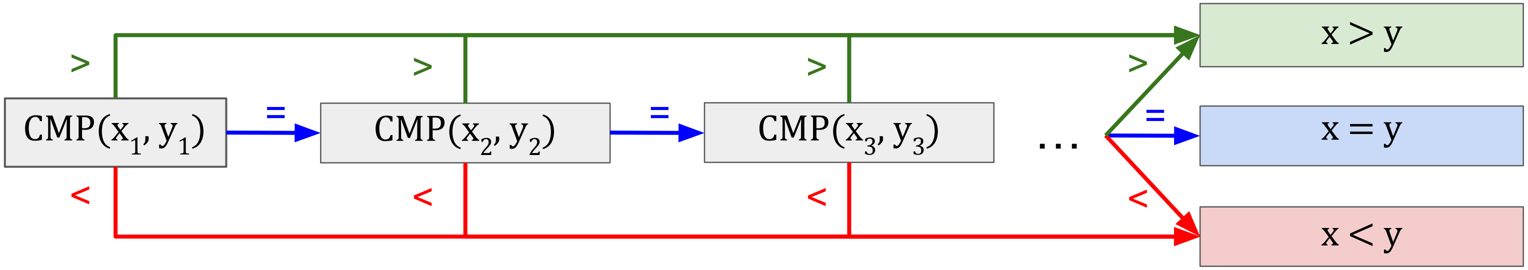
$\text{bit\_decomp}(x) = x_1x_2\ldots x_n$

$\text{bit\_decomp}(y) = y_1y_2\ldots y_n$



Branching Program for Integer Comparison

Comparing x and y.

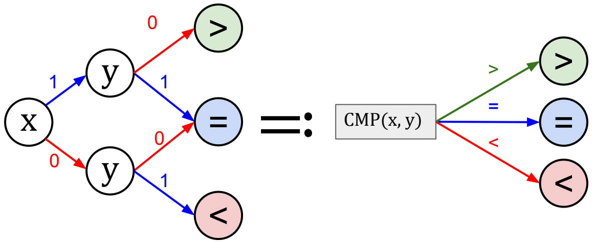


Binary decompositions

$\mathrm{bit\_decomp}(x) = x_1 x_2 \ldots x_n$

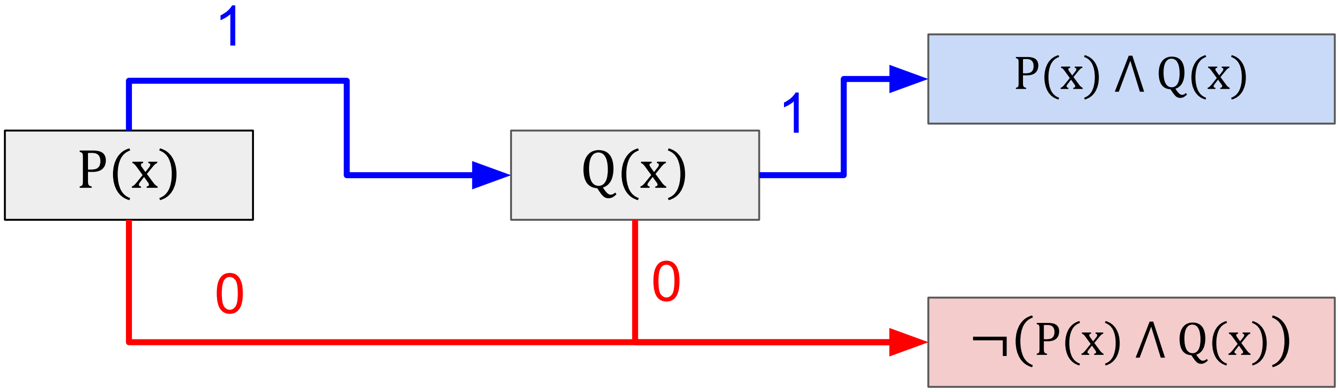
$\mathrm{bit\_decomp}(y) = y_1 y_2 \ldots y_n$

Concrete Cost: $3n$ RMS multiplications



Logical conjunctions

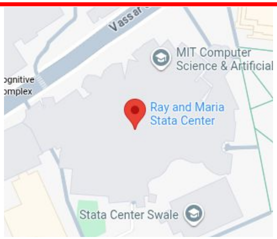
Compute $P(x) \wedge Q(x)$ given branching programs that compute P and Q:

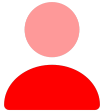


$$\mathrm{Size}(P(x) \wedge Q(x)) = \mathrm{Size}(P(x)) + \mathrm{Size}(Q(x))$$

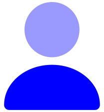
Implementing geolocation-based key exchange

$\vec{x} = (x_1, x_2)$

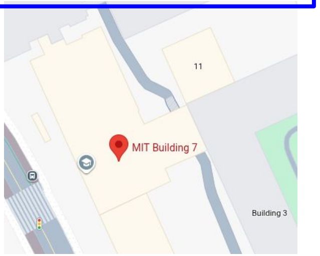




$\xrightarrow{\text{pe}_A}$ $\xleftarrow{\text{pe}_B}$



$\vec{y} = (y_1, y_2)$



Distance Policy: P

$$P(\vec{x}, \vec{y}) = 0 \iff \|\vec{x} - \vec{y}\|_\infty \leq d$$

$$P(\vec{x}, \vec{y}) = 0 \iff \max(|x_1 - y_1|, |x_2 - y_2|) \leq d$$

$$\iff \begin{cases} |x_1 - y_1| \leq d \\ |x_2 - y_2| \leq d \end{cases}$$

$$\iff \begin{cases} \boxed{y_1 - d} \leq \boxed{x_1} \leq \boxed{y_1 + d} \\ \boxed{y_2 - d} \leq \boxed{x_2} \leq \boxed{y_2 + d} \end{cases}$$

Reduces the problem to
four integer comparisons
on
inputs known to some party

Concrete Cost: $4 \times 3\mathrm{n} = \mathbf{12\ n}$ RMS multiplications

Extension: “multi-factor” key exchange

Can use logical conjunctions to combine geolocation and passphrase

Roadmap

1. Overview of our work
2. MKHSS optimizations
3. Non-interactive conditional key exchange optimizations
4. Useful instantiations of key exchange
    a. Fuzzy PAKE
    b. Geolocation-based key exchange
5. Performance evaluation
6. Future works and conclusion

Parameters

Magnitude bound on HSS values: $\mathrm{B} = 1$
(sufficient for all of our key exchange applications, which only uses bits)

Security parameter: $\lambda = 128$

Size of modulus N: 3072 bits, sufficient for 128 bits of security

Runtime: Multi-Key HSS encode

Parameter: $\mathrm{B} = 1$ (sufficient for all of our key exchange applications)
Recall this means all intermediate HSS values are in $\{-1,0,1\}$

Procedure	Our runtime (ms)	Baseline runtime (ms)	Our speedup
KeyGen	7.0	104.5	15×
$\mathrm{EncodeInput}$	2.6	10.3	4.0×

Time for each party to encode input $\mathbf{x} = (\mathrm{x}_1, \ldots, \mathrm{x}_\mathrm{n}) \in [-\mathrm{B}, \mathrm{B}]^\mathrm{n}$:
$\mathrm{Time(KeyGen)} + \mathrm{n} \cdot \mathrm{Time(EncodeInput)}$

Runtime: Multi-Key HSS evaluate

Parameter: $\mathrm{B} = 1$

Procedure	Our runtime (ms)	Baseline runtime (ms)	Our speedup
Init (Alice)	4.4	96.4	22×
Init (Bob)	3.1	50.3	16×
$\mathrm{SyncSelfShare}$	1.3	5.2	4×
$\mathrm{SyncOtherShare}$	1.8	61.8	35×
RMS: Addition	9.5×10^{-3}	38.3×10^{-3}	4.0×
RMS: Multiplication	**5.0**	**224.6**	**45×**

Time to run RMS program $\mathrm{f(x,y)}$: $\mathrm{Time(Init)} + \mathrm{n} \cdot \mathrm{Time(SyncSelfShare)} + \mathrm{m} \cdot \mathrm{Time(SyncSelfShare)}$ + RMS operations

Own input x $= (\mathrm{x}_1, \ldots, \mathrm{x}_\mathrm{n}) \in [-\mathrm{B}, \mathrm{B}]^\mathrm{n}$

Other party's input y $= (\mathrm{y}_1, \ldots, \mathrm{y}_\mathrm{m}) \in [-\mathrm{B}, \mathrm{B}]^\mathrm{m}$

Communication: Multi-Key HSS (continued)

Parameter: $\mathrm{B} = 1$

Data	Size (kB)	Size (kB)	Our saving
Transmitted public key	3.1	6.2	2×
Transmitted input share	1.5	4.6	3×

Communication requirement for one party in MKHSS for input $\mathbf{x} = (x_1, \ldots, x_n) \in [-B, B]^n$:

$$\mathrm{Size(pk)} + \mathrm{n} \cdot \mathrm{Size(InputShare)}$$

Communication: Multi-Key HSS

Parameter: $\mathrm{B} = 1$

Data	Size (kB)	Size (kB)	Our saving
Transmitted public key	3.1	6.2	2×
Transmitted input share	1.5	4.6	3×

The 3× reduction comes from a simplification of input shares

Our Input Share

$$\mathsf{Enc}_{\mathsf{pk}}(x \cdot s) \in \mathbb{Z}_{N^2}^*$$

Baseline Input Share

$$\begin{pmatrix} \mathsf{Enc}'_{\mathsf{pk}}(x \cdot s) \\ \mathsf{Enc}_{\mathsf{pk}}(x) \end{pmatrix} \in \mathbb{Z}_{N^4}^* \times \mathbb{Z}_{N^2}^*$$

Bit Length $2\log_2(N)$ $6\log_2(N)$

Concrete Performance: Fuzzy PAKE Runtime

# chars in password	Bits per char	# typos permitted	Our runtime (sec)	Baseline runtime* (sec)	Our speedup
72	5	2	7.56	252 **(~4 mins)**	33×
80	16	1	19.7	678 **(~11 mins)**	34×
120	8	3	27.5	920 **(~15 mins)**	33×

*: We gave the baseline an advantage by allowing it to use the random-oracle-based construction of key exchange from MKHSS.

Thus the difference is solely due to MKHSS speedups.

Evaluation: Fuzzy PAKE runtime

# chars in password	Bits per char	# typos permitted	Our runtime (sec)	Baseline runtime* (sec)	Our speedup
72	5	2	7.56	252 **(~4 mins)**	33×
80	16	1	19.7	678 **(~11 mins)**	34×
120	8	3	27.5	920 **(~15 mins)**	33×

Further speedup possible with AVX512

[Langowski-Devadas'25]

Our runtime with AVX512 (sec)
3.17
8.47
11.7

Evaluation: Fuzzy PAKE communication

# chars in password	Bits per char	# typos permitted	Our communication cost (MB)	Baseline communication cost (MB)	Our savings
72	5	2	1.1	3.3	**3×**
80	16	1	3.9	11.8	**3×**
120	8	3	3.0	8.9	**3×**

Our **3×** reduction comes from a reduction in the size of input shares of MKHSS

Runtime: Geolocation-Based Key Exchange

# dimensions for coordinate	Precision of coordinate (bits)	Our runtime (sec)	Baseline* runtime (sec)	Our speedup
2	32	1.65	54.1	33×
3	48	3.63	122	33×
4	64	6.46	216	33×

Communication: Geolocation-Based Key Exchange

# dimensions for coordinate	Precision of coordinate (bits)	Our communication cost (kB)	Baseline communication cost (kB)	Our savings
2	32	301.1	897.2	3×
3	48	669.8	2003.1	3×
4	64	1185.9	3551.4	3×

Roadmap

1. Overview of our work
2. MKHSS optimizations
3. Non-interactive conditional key exchange optimizations
4. Useful instantiations of key exchange
    a. Fuzzy PAKE
    b. Geolocation-based key exchange
5. Performance evaluation
6. Future works and conclusion

Future directions

- Add malicious security to key exchange with minimal performance overhead?
  - This can be done generically with zero knowledge proofs
- Application of our structural simplification to other HSS protocols? Recall:
  - $(\cancel{\mathrm{Enc}(x)}, \mathrm{Enc}(x \cdot s)) \rightarrow \mathrm{Enc}(x \cdot s)$
  - $(\cancel{\langle x \rangle}_{\sigma}, \langle x \cdot s \rangle_{\sigma}) \rightarrow \langle x \cdot s \rangle_{\sigma}$
- Other useful predicates for key exchange?
- Other practical applications of MKHSS?

Thank you!

Paper (Lali's talk): https://ia.cr/2025/094

Paper (Kevin's talk): https://ia.cr/2025/1803

Code: https://github.com/kevin-he-01/mkhss

Icons credit: svgrepo.com, flaticon.com, devil horn icon created by freeer, bulletin board icon created by Creatype.

References

[ADOS’22] D. Abram, I. Damgård, C. Orlandi, and P. Scholl. An algebraic framework for silent preprocessing with trustless setup and active security.

[BCGIO’17] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, and M. Orrù. Homomorphic secret sharing: Optimizations and applications.

[BCP’03] E. Bresson, D. Catalano, and D. Pointcheval. A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications.

[BGI’16] E. Boyle, N. Gilboa, and Y. Ishai. Breaking the circuit size barrier for secure computation under DDH.

[CDHJS’25] G. Couteau, L. Devadas, A. Hegde, A. Jain, and S. Servan-Schreiber. Multi-key homomorphic secret sharing.

[DH’76] W. Diffie and M. E. Hellman. New directions in cryptography.

[DHPRY’18] P.-A. Dupont, J. Hesse, D. Pointcheval, L. Reyzin, and S. Yakoubov. Fuzzy password-authenticated key exchange.

[DJ’03] I. Damgård and M. Jurik. A length-flexible threshold cryptosystem with applications.

[FHKP’13] E. S. V. Freire, D. Hofheinz, E. Kiltz, and K. G. Paterson. Non-interactive key exchange.

[Langowski-Devadas’25] S. Langowski and S. Devadas. Efficient modular multiplication using vector instructions on commodity hardware.